University of Thessaly

PhD Dissertation

Exploiting Intrinsic Hardware Guardbands and Software Heterogeneity to Improve the Energy Efficiency of Computing Systems

---

Author:
Panagiotis Koutsovasilis

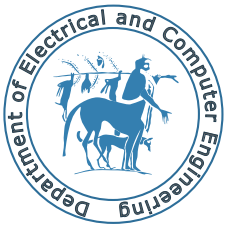
Supervisor:
Christos D. Antonopoulos

Advising committee:
Christos D. Antonopoulos,
Nikolaos Bellas,
Spyros Lalis

A dissertation submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

to the

Department of Electrical and Computer Engineering



March 25, 2020

Πανεπιστήμιο Θεσσαλίας

Διδακτορική Διατριβή

Αξιοποίηση των Εγγενών Περιθωρίων Προστασίας του Υλικού και της Εγγενούς Ετερογένειας του Λογισμικού για τη Βελτίωση της Ενεργειακής Αποδοτικότητας των Υπολογιστικών Συστημάτων

Συγγραφέας:

Παναγιώτης Κουτσοβασίλης

Επιβλέπων:

Χρήστος Δ. Αντωνόπουλος

Συμβουλευτική επιτροπή:

Χρήστος Δ. Αντωνόπουλος,

Νικόλαος Μπέλλας,

Σπύρος Λάλης

Η διατριβή υποβλήθηκε για την εκπλήρωση των απαιτήσεων για την απονομή Διδακτορικού Διπλώματος

στο

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

25 Μαρτίου 2020

Committee

Christos D. Antonopoulos Associate Professor Department of Electrical and Computer Engineering, University of Thessaly Advisor
Nikolaos Bellas Associate Professor Department of Electrical and Computer Engineering, University of Thessaly Advising Committee Member
Dimitris Gizopoulos Professor Department of Informatics and Telecommunications, University of Athens External Committee Member
George Karakonstantis Assistant Professor Electronics, Electrical Engineering and Computer Science, Queen's University of Belfast External Committee Member
Georgios Keramidas Assistant Professor Department of Informatics, Aristotle University of Thessaloniki External Committee Member
Nectarios Koziris Professor Electrical and Computer Engineering, National Technical University of Athens External Committee Member
Spyros Lalis Associate Professor Department of Electrical and Computer Engineering, University of Thessaly Advising Committee Member

“The reality we can put into words is never reality itself.”

Werner Heisenberg

Abstract

Panagiotis Koutsovasilis

Exploiting Intrinsic Hardware Guardbands and Software Heterogeneity to Improve the Energy Efficiency of Computing Systems

A key challenge for both current- and next-generation computing infrastructure deployments is to increase the delivered computational performance within stringent power budget constraints due to power delivery, cooling and cost-related concerns. Techniques such as transistor shrinking, frequency scaling, and parallelism exploitation have contributed to shaping the power efficiency envelope of today's computing systems. However, all of the aforementioned practices are bound to hit the power wall of CMOS technology. Another approach towards power efficiency is hardware-level specialization. Heterogeneous computing combines different architectures, each appropriate for specific computational patterns, within the same system. A typical example is General Purpose programming on Graphics Processing Units (GPGPU), which exploits GPUs to efficiently execute certain classes of embarrassingly parallel computations. Heterogeneous computing, though, comes at the expense of significantly increased programmer effort.

At the same time, chip manufacturers are introducing redundancy at various levels of CPU design to guarantee fault-free operation, even for worst case combinations of non-idealities in process variation and system operating conditions. This redundancy partly translates to CPUs operating at a higher voltage than what is strictly required, in the form of voltage margins. However, these worst case scenarios may appear only rarely or even not at all during the lifecycle of a given machine. Consequently, the operating voltage setting is overly pessimistic for typical operating conditions and leads to excessive power dissipation, which hinders the effort towards improving the power efficiency of computing infrastructures.

On the software side, programmers very often write code that solves problems at a significantly higher accuracy than actually required to meet the Quality of Service (QoS) requirements of the application. In the same vein, all parts of the code are treated as equally important, despite the fact that their contribution to the quality of

the end result may vary significantly. As a result, developers tend to write programs that “over-compute”, a practice which results in lower energy efficiency.

The quest for more energy-efficient systems may necessitate a transition from CMOS towards a more efficient semiconductor technology. However, until a new, commercially viable candidate technology appears, optimizing the energy efficiency of systems based on CMOS technology is a worthy undertaking. In this dissertation we focus on the exploitation of the intrinsic hardware guardbands and software heterogeneity. We design and develop mechanisms that reduce the CPU operating voltage and, thus, minimize the CPU power footprint by considering and exploiting the computational characteristics of the executed workload. Besides improving energy/power efficiency, these mechanisms can also be used to mitigate performance penalties induced on a power-constrained platform. We also evaluate the trade-off between quality of results and energy efficiency when combining heterogeneous computing with various approximation techniques. More specifically:

Power capping is commonly used to regulate the limited power resources of computing infrastructure deployments. We investigate the impact of reducing CPU voltage margins on the efficiency of software- and hardware-based power capping mechanisms on a variety of commercial, off-the-shelf platforms. We observe that operation at reduced voltage margins can significantly improve the efficiency of existing power capping mechanisms in terms of performance, CPU and system node power, as well as CPU temperature. Based on our investigation, we introduce a CPU power capping mechanism that preserves performance in power-constrained environments by operating the CPU at reduced voltage margins to reduce frequency throttling. We show that CPU power capping at reduced voltage margins results in performance improvement by up to 64% and 24% on average, compared with Intel’s RAPL and Dynamic Frequency Scaling (DFS) mechanisms, respectively. Given that CPU operation at reduced voltage margins may potentially limit the effectiveness of a reliability safeguard introduced by manufacturers, we validate the robustness of our approach with a set of long-running experiments. We also show that significant energy gains can be achieved even when taking into account the cost of checkpointing and recovery in large-scale systems.

Furthermore, we investigate the association of workload characteristics with the extent of exploitable reduction of CPU voltage margins. We introduce a run-time governor that dynamically reduces the supply voltage of modern multicore Intel x86-64 CPUs. Our governor employs a model that takes as input a set of performance metrics which are directly measurable via performance monitoring counters and have high predictive value for the minimum tolerable supply voltage, to dynamically predict and apply the appropriate reduction for the workload at hand. Compared to the

conventional DVFS governor of Intel x86-64 CPUs, our approach achieves significant energy savings, up to 42% and 34%, respectively, for two off-the-shelf processor families.

The quantification and exploitation of CPU voltage margins requires long experimental campaigns, involving multiple systems, multiple configurations and numerous different workloads. Moreover, these experimental campaigns have different targets: characterization of CPU voltage margins, data collection, validation and experimental evaluation of the proposed techniques. Motivated by the complexity of these experimental campaigns, we developed a framework that significantly simplifies the configuration and automates the execution of such experimental campaigns. It supports multiple execution modes, namely OS-controlled and bare-metal execution, it can recover from system crashes due to aggressively reduced voltage margins, and can detect symptoms of erratic workload behavior in the form of Machine Check Exceptions (MCEs), system log entries, or even Silent Data Corruptions (SDCs) if the correct output is supplied as a reference.

Finally, we investigate whether the combination of heterogeneous and approximate computing can yield favorable solutions in the energy efficiency vs. quality of results tradeoff. Approximate computing minimizes the energy footprint of applications at the expense of output quality. More specifically, under the premise that not all parts or execution phases of a program affect the quality of its output equally, developers can introduce approximations in less significant parts of their code in an educated manner, in order to improve the energy efficiency of code execution. We experiment with a set of applications from different domains, with diverse characteristics. We have modified the applications to exploit both heterogeneity and approximations. We evaluate them on heterogeneous platforms (comprising of CPUs and GPUs) and quantify the isolated and combined effect of heterogeneous and approximate computing. Our results show that heterogeneous and approximate computing, independently, result in significant energy gains. Moreover, they can be combined to drastically minimize the energy footprint of executions by up to 94%.

Περίληψη

Παναγιώτης Κουτσοβασίλης

Αξιοποίηση των Εγγενών Περιθωρίων Προστασίας του Υλικού και της Εγγενούς Ετερογένειας του Λογισμικού για τη Βελτίωση της Ενεργειακής Αποδοτικότητας των Υπολογιστικών Συστημάτων

Μία βασική πρόκληση για τη σχεδίαση υπολογιστικών συστημάτων στις μέρες μας, αλλά και στο μέλλον, είναι η αύξηση της επεξεργαστικής ικανότητας εντός αυστηρών περιορισμών κατανάλωσης ισχύος, λόγω παραγόντων που σχετίζονται με το κόστος λειτουργίας και ψύξης, αλλά και τεχνικούς περιορισμούς τροφοδοσίας. Τεχνικές όπως η μείωση των διαστάσεων των τρανζίστορ, η διαχείριση της συχνότητας λειτουργίας των μονάδων επεξεργασίας και ο παραλληλισμός έχουν ήδη αξιοποιηθεί ώστε να διαμορφωθεί το τρέχον επίπεδο αποδοτικότητας ισχύος των υπολογιστικών συστημάτων. Όμως όλες οι προαναφερθείσες πρακτικές δεν μπορούν να προσφέρουν σημαντικές περαιτέρω βελτιώσεις. Μια εναλλακτική προσέγγιση για τη βελτίωση της ενεργειακής αποδοτικότητας τέτοιων συστημάτων είναι η εξειδίκευση σε επίπεδο υλικού. Στις ετερογενείς αρχιτεκτονικές συνυπάρχουν διαφορετικά είδη επεξεργαστικών μονάδων, με διαφορετικά χαρακτηριστικά αποδοτικότητας, καθεμιά κατάλληλη για συγκεκριμένα υπολογιστικά μοτίβα. Ένα τυπικό παράδειγμα είναι η υπολογιστική γενικού σκοπού με χρήση καρτών γραφικών (GPUs) – GPGPU – στα πλαίσια της οποίας αξιοποιούνται κάρτες γραφικών για την εκτέλεση μαζικά παραλληλοποιήσιμων υπολογισμών με συγκεκριμένα χαρακτηριστικά. Ωστόσο, η αξιοποίηση της ετερογένειας προϋποθέτει σημαντική επιπλέον προγραμματιστική προσπάθεια κατά το χρόνο ανάπτυξης του λογισμικού.

Την ίδια στιγμή, οι κατασκευαστές υλικού ενσωματώνουν πλεονασμό σε πολλαπλά επίπεδα κατά τη διάρκεια της σχεδίασης επεξεργαστών, ώστε να εγγυηθούν ορθή εκτέλεση ακόμα και υπό δυσμενείς συνδυασμούς παραμέτρων που οφείλονται, μεταξύ άλλων, στην εγγενή κατασκευαστική ανομοιογένεια των τρανζίστορ και στις συνθήκες λειτουργίας του συστήματος. Ένας τέτοιος μηχανισμός είναι η εφαρμογή επαυξημένης τάσης τροφοδοσίας του επεξεργαστή σε σχέση με αυτή που πραγματικά

απαιτείται (περιθώριο τάσης τροφοδοσίας). Ωστόσο, ο συνδυασμός ακραία δυσμενών παραμέτρων λειτουργίας είναι εξαιρετικά σπάνιος ή μπορεί να μην εμφανιστεί ποτέ κατά τη διάρκεια του κύκλου ζωής των επεξεργαστών. Συνακόλουθα, η διαχείριση της τάσης τροφοδοσίας είναι υπερβολικά απαισιόδοξη για τις τυπικές συνθήκες λειτουργίας και οδηγεί σε αχρείαστα υψηλή κατανάλωση ισχύος, λειτουργώντας ως εμπόδιο στην προσπάθεια βελτίωσης της ενεργειακής αποδοτικότητας των συστημάτων.

Στο επίπεδο του λογισμικού, οι προγραμματιστές πολύ συχνά γράφουν κώδικα που επιλύει προβλήματα με σημαντικά υψηλότερη ακρίβεια από αυτή που πράγματι απαιτείται για την κάλυψη των απαιτήσεων ποιότητας της εφαρμογής. Στο ίδιο πλαίσιο, όλα τα τμήματα του κώδικα αντιμετωπίζονται ως εξίσου σημαντικά, παρόλο που η συνεισφορά τους στην ποιότητα του τελικού αποτελέσματος μπορεί να διαφέρει σημαντικά. Κατά συνέπεια τείνουμε να υπερ-υπολογίζουμε και η τάση αυτή επίσης λειτουργεί επιβαρυντικά ως προς την ενεργειακή αποδοτικότητα.

Η ανάγκη για περισσότερο ενεργειακά αποδοτικά συστήματα ενδέχεται να ενθαρρύνει τη μετάβαση από την τεχνολογία CMOS προς μια αποδοτικότερη τεχνολογία σχεδίασης ημιαγωγών. Ωστόσο, μέχρι να εμφανιστεί μια νέα, εμπορικά βιώσιμη υποψήφια τεχνολογία, η βελτιστοποίηση της ενεργειακής αποδοτικότητας των συστημάτων που βασίζονται στην τεχνολογία CMOS αποτελεί σημαντική διέξοδο και αντικείμενο έρευνας. Στην παρούσα διδακτορική διατριβή διερευνώνται τρόποι βελτιστοποίησης της ενεργειακής αποδοτικότητας υπολογιστικών συστημάτων αξιοποιώντας την μείωση των επαυξημένων ορίων λειτουργίας υλικού και την ετερογένεια του υλικού και του λογισμικού. Αναπτύσσουμε μηχανισμούς που μειώνουν την τάση λειτουργίας του επεξεργαστή, και συνακόλουθα το ενεργειακό του αποτύπωμα, αναλόγως των υπολογιστικών χαρακτηριστικών των εκτελούμενων υπολογιστικών έργων. Πέραν της βελτίωσης της ενεργειακής αποδοτικότητας, αυτοί οι μηχανισμοί μπορούν επίσης να χρησιμοποιηθούν για την ελαχιστοποίηση της επίπτωσης στην επίδοση των συστημάτων, όταν αυτά εξαναγκάζονται να λειτουργούν υπό περιορισμούς κατανάλωσης ισχύος. Επίσης, αξιολογούμε τη σχέση μεταξύ της ενεργειακής αποδοτικότητας και της ποιότητας αποτελεσμάτων όταν συνδυάζεται η χρήση ετερογενών συστημάτων με τεχνικές προσεγγιστικού υπολογισμού. Πιο συγκεκριμένα:

Η επιβολή ενός μέγιστου ορίου κατανάλωσης ισχύος των επεξεργαστών (power capping) αποτελεί μια καθιερωμένη τεχνική για τη διαχείριση των περιορισμένων πόρων τροφοδοσίας υπολογιστικών υποδομών. Στη διατριβή μελετάμε τον αντίκτυπο της λελογισμένης μείωσης του επαυξημένου ορίου τάσης τροφοδοσίας των επεξεργαστών στην αποδοτικότητα μηχανισμών power capping που υλοποιούνται σε επίπεδο είτε λογισμικού, είτε υλικού. Πειραματιζόμαστε με πολλαπλές, εμπορικά διαθέσιμες αρχιτεκτονικές επεξεργαστών. Παρατηρούμε ότι η μείωση της πλεονάζουσας

τάσης λειτουργίας οδηγεί σε βελτίωση της επίδοσης, της κατανάλωσης ενέργειας σε επίπεδο επεξεργαστή και κόμβου και της θερμοκρασίας του επεξεργαστή. Βασισμένοι στην παραπάνω μελέτη, εισάγουμε έναν νέο μηχανισμό περιορισμού της ισχύος λειτουργίας του επεξεργαστή, που αξιοποιεί την μείωση της πλεονάζουσας τάσης τροφοδοσίας και προσφέρει υψηλότερη επίδοση έως 64% και 24% κατά μέσο όρο, σε σχέση με καθιερωμένους μηχανισμούς όπως το Intel RAPL και η δυναμική κλιμάκωση συχνότητας, αντίστοιχα. Ωστόσο, η λειτουργία του επεξεργαστή με μειωμένα περιθώρια τάσης τροφοδοσίας ενδέχεται δυνητικά να μειώσει την αξιοπιστία λειτουργίας του συστήματος. Για το λόγο αυτό, επικυρώνουμε τον μηχανισμό μας μέσω μιας σειράς μακρόχρονων πειραμάτων. Επιπλέον δείχνουμε πως η λειτουργία του επεξεργαστή με μειωμένα περιθώρια τάσης τροφοδοσίας παραμένει κερδοφόρα, ως προς την ενεργειακή κατανάλωση, ακόμα και σε συστήματα μεγάλης κλίμακας και ακόμη και όταν συνυπολογίσουμε την επιπλέον επιβάρυνση, λόγω της δυνητικά μειωμένης αξιοπιστίας, μηχανισμών για την αντιμετώπιση πιθανών σφαλμάτων, όπως η χρήση checkpointing / αποκατάστασης.

Επιπρόσθετα, μελετάμε τη συσχέτιση των χαρακτηριστικών του υπολογιστικού έργου με το εύρος της αξιοποιήσιμης μείωσης της πλεονάζουσας τάσης τροφοδοσίας του επεξεργαστή. Πιο συγκεκριμένα εισάγουμε μηχανισμό ο οποίος μειώνει δυναμικά την τάση τροφοδοσίας σε επεξεργαστές Intel x86-64. Ο μηχανισμός μας βασίζεται σε μοντέλο το οποίο δέχεται ως είσοδο – κατά το χρόνο εκτέλεσης – ποσοτικές πληροφορίες για την αλληλεπίδραση του λογισμικού που εκτελείται με το υποκείμενο υλικό. Οι πληροφορίες αυτές προέρχονται από τους μετρητές συμβάντων που είναι διαθέσιμοι στους επεξεργαστές Intel και έχουν προγνωστική αξία για την ελάχιστη ανεκτή τάση τροφοδοσίας, η οποία αποτελεί και την έξοδο του μοντέλου. Ο μηχανισμός μας συνεπώς υπολογίζει και επιβάλλει δυναμικά την κατάλληλη κάθε στιγμη μείωση τάσης τροφοδοσίας για το υπολογιστικό έργο που εκτελείται. Συγκριτικά με καθιερωμένους μηχανισμούς που διαχειρίζονται τις παραμέτρους λειτουργίας (τάση και συχνότητα) του επεξεργαστή, ο μηχανισμός μας επιτυγχάνει έως 42% και 34% αντίστοιχα μείωση της ενεργειακής κατανάλωσης για δύο εμπορικά διαθέσιμες οικογένειες επεξεργαστών Intel.

Η ποσοτικοποίηση και η αξιοποίηση της μείωσης των περιθωρίων τάσης τροφοδοσίας του επεξεργαστή απαιτεί ευρύ, μακρόχρονο πειραματισμό, με πολλαπλά συστήματα και πολυάριθμες εφαρμογές. Επιπλέον, οι ακολουθίες πειραμάτων ενδέχεται να έχουν διαφορετικούς στόχους: την ποσοτικοποίηση των περιθωρίων τάσης τροφοδοσίας του επεξεργαστή, τη συλλογή δεδομένων (profiling), την επικύρωση και την πειραματική αξιολόγηση των προτεινόμενων μηχανισμών. Ορμώμενοι από την εγγενή πολυπλοκότητα που έχουν τέτοιες πειραματικές καμπάνιες, σχεδιάζουμε και υλοποιούμε υποδομή που απλοποιεί σημαντικά τον ορισμό και αυτοματοποιεί

την εκτέλεση τέτοιων ακολουθιών πειραμάτων. Η υποδομή μας υποστηρίζει πολλαπλούς τρόπους εκτέλεσης εφαρμογών, είτε πάνω από λειτουργικό σύστημα, είτε απευθείας πάνω από το υλικό, μπορεί να ανακάμψει από κατάρρευση του συστήματος λόγω επιθετικής μείωσης των περιθωρίων τάσης τροφοδοσίας και μπορεί να ανιχνεύσει συμπτώματα ασταθούς συμπεριφοράς της εκάστοτε εφαρμογής ή του συστήματος ελέγχοντας για λάθη Machine Check Exceptions (MCEs) και Silent Data Corruptions (SDCs), τα τελευταία εφόσον η αναμενόμενη σωστή έξοδος είναι διαθέσιμη.

Τέλος, διερευνούμε κατά πόσο ο συνδυασμός ετερογένειας του υλικού και η προσεγγιστική υπολογιστική μπορεί να αποφέρει επιθυμητές λύσεις ως προς τη σχέση ενεργειακής αποδοτικότητας και μείωσης ποιότητας των αποτελεσμάτων. Η προσεγγιστική υπολογιστική ελαχιστοποιεί το ενεργειακό αποτύπωμα των εφαρμογών σε βάρος της ποιότητας των αποτελεσμάτων. Πιο συγκεκριμένα, δεδομένου ότι όλα τα τμήματα ή οι φάσεις εκτέλεσης μιας εφαρμογής δεν επηρεάζουν εξίσου την ποιότητα των αποτελεσμάτων, οι προγραμματιστές μπορούν να αξιοποιήσουν, με στοχευμένο τρόπο, προσεγγιστικές τεχνικές σε λιγότερο σημαντικά τμήματα του κώδικα τους, προκειμένου να βελτιωθεί η ενεργειακή αποδοτικότητα του εκτελούμενου κώδικα. Πειραματιζόμαστε με ένα σύνολο εφαρμογών με διαφορετικά χαρακτηριστικά, από διαφορετικούς τομείς της επιστήμης και της μηχανικής. Οι εφαρμογές τροποποιήθηκαν ώστε να αξιοποιούν τόσο την ετερογένεια υλικού, όσο και την προσεγγιστική υπολογιστική. Η αξιολόγηση πραγματοποιείται σε ετερογενείς πλατφόρμες (που περιλαμβάνουν επεξεργαστές και κάρτες γραφικών) με στόχο την ποσοτικοποίηση του επιμέρους και του συνδυασμένου οφέλους λόγω της χρήσης ετερογενών συστημάτων και προσεγγιστικής υπολογιστικής. Τα αποτελέσματά μας δείχνουν πως η αξιοποίηση της ετερογένειας του υλικού και του λογισμικού (μέσω προσεγγίσεων) ανεξάρτητα έχει ως αποτέλεσμα τη μείωση της κατανάλωσης ενέργειας. Επιπλέον, μπορούν να συνδυαστούν, ώστε να μειώσουν δραστικά το ενεργειακό αποτύπωμα της εκτέλεσης λογισμικού έως και κατά 94%.

Acknowledgements

This dissertation is the result of research work conducted while I was pursuing my PhD degree in the Department of Electrical and Computing Engineering of University of Thessaly in Greece.

First and foremost, I would like to thank my mentors, Professors Christos D. Antonopoulos, Nikolaos Bellas, and Spyros Lalis from the University of Thessaly. They have been exceptionally good at guiding me during my initial steps, throughout my MSc and PhD. Without their guidance and mentoring none of this work would be possible. They were always available to discuss and provide constructive criticism.

I would also like to thank my family. Especially, I owe to my father, Spyros, for teaching me to always pursue my dreams and my mother, Helen, for her unconditional love and support all along my academic pursuits. Also, my friends will always have a special place in my heart because they were always there during good times, and bad times to support me with patience and love.

Furthermore, special thanks to my friends and colleagues Kalogirou Christos and Maroudas Manolis with whom I shared, pretty much all of my research career thus far. Our joined research efforts and stimulating discussions were very educating.

Last but not least, I acknowledge the funding agencies which made this research possible with their financial support. These include the European Commission through the UniServer H2020 and SCoRPiO FP7 projects, and the European Social Fund and Greek national resources through the Centaurus project (Aristeia II action).

Contents

List of Figures

List of Tables

Dedicated to my family and
friends

Chapter 1

Introduction

The Internet is on the verge of a turning point due to the ever-increasing Internet-connected intelligent devices that in the upcoming decade will be in the orders of tens of billions forming the Internet of Things (IoT). Each intelligent device is pushing a little data to the Internet, and a little data pushed by billions of smart devices in Homes, Cities, and Environments will be aggregated to become Big Data, stored and processed currently in huge central datacenters. These large data sets are becoming a core asset in the economy, fostering new industries, processes, and products and creating significant competitive advantages [1]. Turning this opportunity into products and employment growth critically depends on overcoming a formidable obstacle: harnessing efficiently the imminent data flood in the Future Internet. This is contingent on improving the performance of servers that run internet/cloud-based services while reducing their power consumption. This is very important for reducing the running costs in a server farm that supports today's datacenters and cloud providers, while at the same time it enables the placement of servers co-located with the origin of the data (e.g. sensors) where power is limited [2].

As a consequence, current and next-generation devices and systems must evolve to (a) operate on completely new principles, and (b) support completely new architectures.

1.1 Problem

Modern datacenters and high-performance computing (HPC) systems are expected to operate under a tight power budget due to cost, power delivery, and cooling limitations. This is a challenging undertaking, as in the past decade power and cooling costs have doubled [3]. In particular, CPUs account for up to 60% of the total power consumption of compute nodes [4], whereas datacenter cooling has a power footprint equal to that of the compute infrastructure. An important exercise for both the datacenter and HPC domains is to design and deploy techniques that optimize throughput/performance in a power-constrained environment.

However, aggressive CMOS technology scaling into lower nanometer geometries has led to the variability of transistor characteristics. Traditionally, techniques for dealing with transistor variability involve extra provisioning in logic and memory circuits, in the form of increased voltage margins, reduced operating frequencies and error correction circuitry. Such voltage margins are specified at design time by taking into account the implementation technology, power budget, worst case timing paths, operating conditions, and fabrication process variations.

These voltage margins lead to significant power overheads, which conflicts with the challenges of limiting power dissipation. The average power overhead of CPU voltage margins can be in the order of 35% [5], yet most of the time these margins are excessive and translate to unnecessary power overhead, as the worst-case combinations of adverse conditions that were considered at design time may appear only rarely or even not at all during the life cycle of a given processor. Providing an end-to-end approach that effectively reduces these margins, could have a significant impact on such systems, as the resulting energy gains would enable the utilization of extra resources to support additional parallelism and increased computational capacity. A critical challenge though is to reduce the margins as much as possible, yet without compromising the reliability of the system.

Another hardware/software approach to improve performance per Watt, that does not affect the reliability of the system but requires extra programmer effort, is heterogeneous computing. Heterogeneous computing exploits accelerators which are optimized for efficiently executing certain computational patterns. GPUs are a popular family of accelerators which, in contrast to conventional CPU-like architectures, offer massive, partially asynchronous parallel execution through many computational cores. Although their power footprint is slightly higher than that of a typical CPU, they are superior in terms of performance per Watt. At the expense of a few extra Watts, applications that exhibit sufficient parallelism and GPU-friendly computation and memory access patterns can utilize GPUs to reduce execution time and by extension their energy footprint.

A more aggressive software approach towards improving the energy efficiency of applications, which, similarly to heterogeneous computing requires extra programmer effort, is approximate computing. Computing systems are traditionally engineered and expected to execute programs under the assumption that all computations have the same significance (importance) for the quality of program output. Approximate computing disrupts this rather conservative approach, by relaxing the requirement for fully accurate output, thus enabling a new trade-off among performance, energy footprint and quality of results.

The combination of heterogeneous and approximate computing opens up new

possibilities for power/energy management, energy footprint minimization of applications and hardware classification in terms of energy efficiency. Software can be extended by approximate computing techniques to enable applications that operate under tight energy budgets. Hardware accelerators can be designed to natively support different approximations when instructed to. Large scale datacenters and HPC infrastructure deployments could significantly cut down their power costs, adapt to more stringent power caps and set new standards to metrics such as performance per Watt.

1.2 Motivation

Figure 1.1 depicts the power consumption of *bzip2* from the SPEC CPU2006 benchmark suite [6] for three different execution scenarios: one without a CPU power cap (*dark red* line) and two at a CPU power cap of 20W at nominal (*orange* line) and at reduced CPU voltage margins (*green* line). The experiments have been performed on a Xeon E3-based node which has a Thermal Design Power (TDP) of 80W, and, thus, the 20W power cap is rather restrictive. We concurrently run multiple instances of *bzip2* to fully engage all four cores of the CPU. The horizontal distance between the ending points of the *dark red* and *orange* vertical lines indicate the performance penalty due to CPU power capping. Similarly, the horizontal distance between the ending points of the *orange* and *green* vertical lines represent the performance gains of CPU voltage margins reduction, compared with nominal CPU operation for the same cap.

In the power-constrained execution with nominal CPU operation, the clock frequency is reduced significantly to meet the cap, leading to a 36% drop in performance.

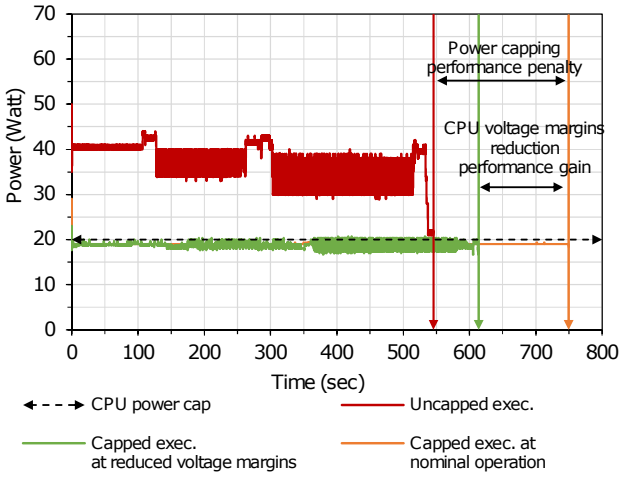


Figure 1.1: Impact of reducing CPU voltage margins on performance when the CPU operates at a power cap.

On the other hand, with CPU operation at reduced voltage margins, the 20W power cap is met without underscaling the frequency as much, and hence, with substantially lower performance degradation. More specifically, for the power-constrained execution at nominal voltage, the CPU frequency is 1.9 GHz on average and for the execution at reduced voltage margins (-170 mV reduction) the average CPU frequency is 2.5 GHz. As a result, we observe 22% improved performance at reduced voltage margins compared with capped execution at nominal voltage. These results demonstrate the opportunity of harnessing voltage margins to meet restrictive power caps without resorting to extreme frequency downscaling.

Furthermore, for certain processor families, CPU voltage margins depend on the computation characteristics of the executing workload. More specifically, the reduction of voltage margins, that does not compromise the system reliability, varies for workloads that exhibit different resource utilization patterns. For example, for the Xeon E3 system of the previous example, we observe that the safe reduction of CPU voltage margins is -192mV and -270mV for h264ref and gromacs applications, respectively, from the SPECCPU 2006 [7] benchmark suite. This divergence demonstrates the opportunity of increasing the energy savings, due to operation at reduced voltage margins, for several applications.

However, as already mentioned CPU operation at reduced voltage margins may affect the system reliability. An approach that does not affect the reliability of the system and offers significant energy savings is the combination of heterogeneous with approximate computing. Approximate computing disrupts the traditional and rather conservative computing approach, by relaxing the requirement for fully accurate output, thus enabling a new trade-off among performance, energy footprint and quality of results. Approximate computing builds upon two observations: (a) several application domains, such as simulations, computer vision, media applications, etc. can tolerate a certain degree of results imprecision, and (b) result quality is not equally affected by all computations of a program; some computations can be approximated or even dropped without heavily penalizing output quality.

As an example, Figure 1.2 presents the output of an application that estimates a dense disparity map of a scene. As shown in the Figure, there are minor differences between the results of the accurate and the approximate versions of the application, because the approximation used lowers the resolution of the sub-regions scanning in the scene. However, the main outputs of the application, namely object positioning, and disparity estimations are still correct. Moreover, with this approximation and by executing the application entirely on a GPU we can achieve sufficient quality of output with 37% improvement in energy consumption, compared with accurate execution of the application on a CPU.

Figure 1.2: Accurate (left) vs Approximate (right) output of a disparity map depth estimation application.

1.3 Contributions

1.3.1 CPU operation at reduced voltage margins

We study the reduction of CPU voltage margins, based on the premise that the worst-case combinations, which were considered at design time and translate to higher CPU operating voltage in the form of voltage guardbands, may appear only rarely or even not at all during the lifecycle of a given processor. On the other hand, the reduction of CPU voltage margins may compromise the reliability of the system. This mandates an extensive experimental and theoretical study that ultimately answers the following questions: "How to exploit the voltage margins reduction ?", "Is the system reliability affected ?", and "How to automate the complex experimentation process ?".

How to exploit the voltage margins reduction?
A critical step in the exploitation of voltage margins is to identify how they can be reduced without compromising, at least observably, system reliability. More specifically, we need to identify how different CPU utilization scenarios affect the amount by which the CPU operating voltage can be reduced. As an example, we need to observe the difference in voltage margins when a workload utilizes all the cores versus a single core of the CPU, as well as, how applications with different computational characteristics affect the CPU operation at reduced voltage margins.

In the context of this dissertation, we investigate all of the aforementioned aspects and present the results for four different commercially available, off-the-shelf, processors. We find that there is a significant amount of divergence in the width of the CPU voltage margins, depending on the microarchitecture of the processor, and there is a substantial differentiation, even for processors of the same architecture.

Based on our findings, which we will present in detail in the following Chapters, we observe that for certain processors, that exhibit wide voltage margins, there is workload dependability of the extent of CPU operating voltage reduction that can be applied without leading to erroneous execution. However, there are processors, with

significantly lower voltage margins, for which the aforementioned effect is almost negligible and completely diminishes when the workload utilizes all the CPU cores. In other words, there is a static portion of CPU voltage margins that can be exploited, but in certain architectures, there is also another dynamic (workload-dependant) portion that can be exploited and leads to greater energy savings. To this end, we present two approaches in the following Sections that exploit appropriately both static and dynamic voltage margins for the corresponding CPUs. More specifically,

- We study the effects of reducing the width of the static CPU voltage margins under power-constrained execution. To the best of our knowledge, this is the first work that studies the interplay between CPU power capping and operation at reduced margins, using multiple evaluation metrics and experimenting across multiple platforms. Also, we experimentally, on real systems, demonstrate that conventional CPU power capping mechanisms can be combined with operation at reduced voltage margins.
- We exploit the workload-dependability of CPU voltage margins by designing an online voltage scaling governor that dynamically adjusts the supply voltage based on a model predicting the extent of exploitable voltage margins. The model is trained using data collected during characterization and profiling experimental campaigns.

Is system reliability affected?

Although, the aforementioned approaches reduce the CPU voltage margins in an educated manner that does not lead to observable erratic behavior, given that the CPU operates outside its nominal, manufacturer-defined operating envelope, the possibility of the CPU accidentally entering an operating region where errors may occur can not be eliminated. Both in current and future large-scale systems reliability is and will remain an inherent, first-class design concern, leading to the implementation and use of fault tolerance mechanisms (such as checkpointing and restart), even for systems operating within their nominal configuration envelope. In order to validate the robustness of our mechanisms for reducing voltage margins, we orchestrate a large-scale validation campaign on 16 systems, which lasted for 23 days. Based on our results, we provide a pessimistic estimation of the new Mean Time Between Failures (MTBF) when the CPU operates at reduced voltage margins and we show that, even when combined with a fault-tolerance mechanism such as checkpointing, the energy gains of voltage margins reduction remain attractive.

How to automate the complex experimentation process?

A necessary undertaking for exploiting CPU voltage margins is quantifying the width of the respective margins. Moreover, mechanisms that operate the systems outside their nominal configuration envelope mandate long, complex validation and evaluation campaigns. As a consequence, there is a need for a framework that supports multiple platforms of different architectures, supports different software stacks – namely OS-controlled and bare-metal execution – can reliably extract the voltage margins of a CPU, and evaluate the performance and robustness of the aforementioned mechanisms. Also, this framework must identify symptoms of errors due to operation at reduced voltage margins, such as Machine Check Exception errors, and Silent Data Corruptions. At the same time, it needs to be resilient to external problems, such as power outages or network failures. We introduce eXtended Margins eXperiment Manager ($(\mathrm{XM})^2$) which automates the evaluation of software on systems operating outside their nominal configuration envelope. Although $(\mathrm{XM})^2$ supports both bare-metal and OS-controlled execution, we will focus on the OS-controlled flavor.

1.3.2 Combining approximate & heterogeneous computing

We study the combination of heterogeneous with approximate computing, based on the premise that specific phases of computation may incur a high performance and energy toll without a significant contribution to the quality of the result. We focus on applications for which the algorithmic logic of at least some of their phases can be efficiently executed on accelerators such as GPUs. Our study ultimately provides answers to the following questions: “Can approximate and heterogeneous computing be combined ?”, and “What is the effect in the energy efficiency vs. quality of results tradeoff ?”.

Can approximate and heterogeneous computing be combined?

There are several applications for which the combination of approximate and heterogeneous computing can yield significant energy consumption savings without significant / non-tolerable effect to the quality of results. For example, applications that simulate systems of bodies, ranging from atoms to stars and galaxies, are a perfect match to apply the combination of heterogeneous with approximate computing. The governing law of such a system is the equation of motion for particles and, thus, the computational phase can be parallelized at the granularity of a particle, enabling the exploitation of a GPU. Moreover, due to the nature of the simulation, not all particles affect with the same significance the motion of other particles. More specifically, particles that are further away, from the particle of interest, have negligible impact on its motion. As a result, an approximation that would result in saving computational

resources with minimal impact to the quality of results is setting a cut-off distance beyond which the interaction of particles is not considered.

What is the effect in the energy efficiency vs. quality of results tradeoff? Effectively combining heterogenous with approximate computing to gracefully trade-off application output quality with energy/performance gains requires a systematic approach to executing programs using the principles heterogeneous and approximate computing. We introduce a set of 7 applications, that are part of AcHEe (Approximate Computing and Heterogeneity for Energy efficiency), modified to exploit both heterogeneity and approximate computing. Our application set is developed using mainly OpenCL nomenclature, therefore it can target any architecture and accelerator device supporting OpenCL. Each application comes with both accurate and approximate implementations of its computationally intensive parts. The latter exploit different types of approximations, carefully chosen to balance between energy efficiency and quality degradation of results. For example, in the case of the DCT MV application we find that the combination of heterogeneous and approximate computing resulted in 98.7% improved energy efficiency compared with a fully accurate execution on a CPU, at the expense of a tolerable quality loss (PSNR lower by 5 dB, from 38 dB to 33 dB).

1.4 Outline

Chapter 2 provides the technical background assumed by the following Chapters.

Chapter 3 introduces our approach for exploiting CPU voltage margins reduction under power-constrained execution. To this end, we present RVSCap, a novel CPU power capping mechanism that operates the CPU at reduced voltage margins, minimizes the performance penalty of power capping at restrictive caps, reduces the power footprint at more generous caps and supports multiple platforms.

Later, in Chapter 4, we introduce xDVS. xDVS is a dynamic voltage scaling governor that exploits the workload dependability of CPU voltage margins. More specifically, we discuss the prediction models of voltage margins reduction and then focus on the design, implementation and evaluation of the xDVSgovernor.

Chapter 5 validates RVSCap and xDVS. We, then, discuss system reliability, fault-tolerance aspects and present the expected energy gains in large-scale deployments where a fault-tolerance mechanism, such as checkpointing, is necessary.

Chapter 6 introduces $(\mathrm{XM})^2$, a framework to automate experimental campaigns for CPU voltage margins characterization, application profiling, xDVS and RVSCap-validation and evaluation.

Then, in Chapter 7, we present our modifications to a set of applications in order to exploit both heterogeneity and approximate computing. In particular, we showcase in practice different approximation techniques for various applications. Moreover, we show that when these approximations are combined with the utilization of accelerators, such as GPUs, the energy efficiency is significantly improved.

Finally, Chapter 8 concludes the dissertation.

Chapter 2

Background

This Chapter outlines the necessary background to assist the reader in following the discussion in the subsequent Chapters, which introduce the contributions of this dissertation.

2.1 Platforms Overview

This Section explains how we adjust the operating points of the CPUs used in the context of this dissertation (Table 2.1), briefly discusses their power-saving modes and outlines their performance monitoring functionality.

2.1.1 Intel-Based systems

Frequency adjustment

Intel CPUs from the Skylake family, and later on, feature the Speedshift [8] mechanism. As shown in Figure 2.1, Speedshift enables fully autonomous control of P-State selection by the CPU, known as Hardware P-States (HWP), supporting a dynamic, wide power range with faster transition times between the minimum and maximum supported CPU frequencies. To this end, for all the CPUs, utilized in the context of

Table 2.1: Characteristics of the four target CPUs.

Characteristic	Xeon E3-1220v5	i7-4790K	X-Gene 2	X-Gene 3
CPU Cores	4	4	8	32
Hardware threads	4	8	8	32
Base clock freq.	3.0GHz	4.0GHz	2.4GHz	3.0GHz
Turbo clock freq.	3.3GHz	4.4GHz	-	-
Lowest clock freq.	0.8GHz	0.8MHz	0.3GHz	0.375GHz
Supply Voltage (V_{dd})	1.15V	1.07V	0.98V	0.87V
Manufacturer	Intel	Intel	APM	APM
Family	Skylake	Haswell	ARMv8	ARMv8
Technology	14nm	22nm	28nm	16nm
Max Performance	4 (issue-slots/cycle)	4 (issue-slots/cycle)	4 (IPC)	4 (IPC)
Freq. Control	HWP	P-State	CPPC	CPPC
TDP (W)	80	88	35	125

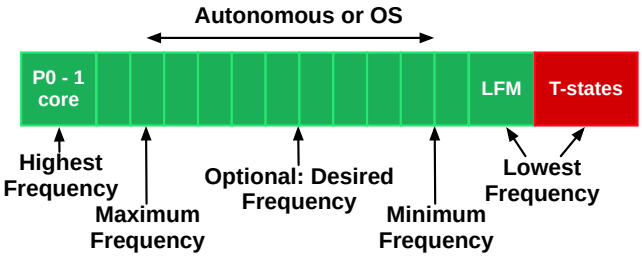


Figure 2.1: Speedshift overview.

this dissertation, all cores are clocked with the same frequency. The OS may disable autonomous scaling by setting the maximum and minimum frequencies of a P-State to the same value. Prior Intel CPUs, before the Skylake family such as the Haswell family, feature a similar mechanism, namely the Intel P-State driver, which features the same logic as Speedshift but is implemented in software.

Supply voltage adjustment

Moreover, the latest generations of Intel CPUs, from the Haswell family and later on, feature the Fully Integrated Voltage Regulator (FIVR) [9] mechanism which selects the optimal supply voltage V_{dd} for CPU cores according to the frequency of the CPU cores and the executing workload (Figure 2.2). More specifically, FIVR operation requires the presence of voltage regulators (VRs) which are separated into two stages, with the first stage of VRs responsible for converting the Power Supply Unit (PSU) or battery voltage (12-20V) to approximately 1.8V and distributing it across the CPU

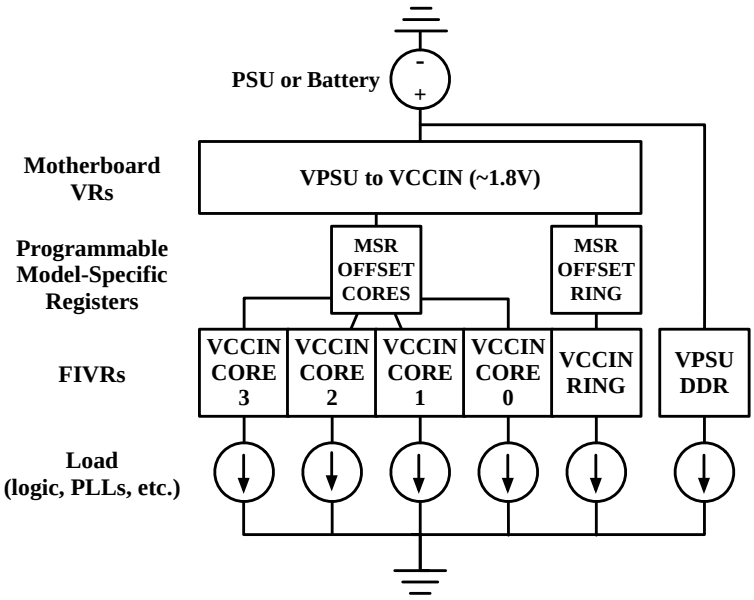


Figure 2.2: Overview of Intel FIVR.

die. The second stage is comprised of inner-chip VRs – the total number of VRs is product dependant – which are 140 MHz synchronous multiphase buck converters with up to 16 phases. Each FIVR is independently programmable to achieve optimal operation of the domain it is powering.

It is possible for the software to alter the supply voltage, as shown in Equation 2.1, by writing an offset value into specific Model-Specific Registers (MSRs). Although there are separate MSRs for the core and uncore components, we empirically observed that the supplied voltage changes only when both registers are set to the same value. Thus, we operate the core and the uncore components with the same voltage offset ($\text{MSR}_{\text{offset}}$). Note that FIVR does not support an independent per-core adjustment of the offset, therefore the offset applies to all CPU cores.

$$V'_{dd} = V_{dd} - MSR_{offset} \tag{2.1}$$

Power and performance monitoring mechanisms

Furthermore, Intel CPUs can save energy when idling, by setting CPU cores in a low-power mode. There are several such modes, or so-called *C-states*, which perform clock and power gating to different units inside the core. C-states are numbered starting from $C0$, the normal CPU operating state in which all CPU modules are powered up and clocked. Higher power-saving states result in an increasing number of circuits and signals being power- or clock-gated, putting the core into a deeper sleep state. The deeper the core sleep state, the higher the performance and energy penalty to revert to $C0$.

Another power-related mechanism in the arsenal of modern Intel CPUs, including the ones used in the context of this dissertation, is the RAPL [10] mechanism. RAPL is a hardware power reporting and capping mechanism that directly observes the power consumption of the CPU. When RAPL enforces a power cap, it samples CPU power consumption within a user-specified time window and solves a linear equation [10, 11] to dynamically decide and apply the most suitable frequency and voltage pair from the DVFS operating points supported by the CPU. The cap enforcement time window can be as narrow as 1 msec.

To quantify the interaction of applications with hardware Intel CPUs have on-chip Performance Monitoring Units (PMUs), used. For the CPUs used in this dissertation, only up to 8 performance counters per core can be monitored simultaneously. When the hyper-threading capability is enabled, the number of available registers is reduced to 4. Exceeding these limits causes the interleaved monitoring of performance counters. Typically, the PMUs are used by profilers to obtain the CPU resource utilization.

2.1.2 ARM-based platforms

Frequency adjustment

Both ARMv8 microprocessors, namely the Applied Micro's X-Gene 2 and X-Gene 3, used in the context of this dissertation, offer high-end processing performance and contain a dedicated subsystem that features Power Management processor (PMpro) which is orchestrated by a Scalable Lightweight Intelligent Management processor (SLIMpro) to enable flexibility in power management, resiliency, and end-to-end security. The dedicated PMpro processor exposes advanced power management capabilities, such as thermal protection, configures system attributes (e.g. regulates supply voltage, etc.) and clock gating mechanisms. Such capabilities are monitored and managed by the SLIMpro which communicates directly with PMpro through an integrated I^2C controller and can be accessed by the Linux kernel.

Both processors support the Collaborative Processor Performance Control (CPPC) power and performance management API as it is defined in ACPI 5.0+ specification. CPPC is a new approach to control the CPU performance using an abstract continuous scale in contrast with the traditional discretized P-state scale. As shown in Figure 2.3, the flow of adjusting the operating frequency involves initially the OS making a change request of the frequency to the platform. The communication between the OS and platform occurs through the Platform Communication Channel (PCC). Afterward, the platform (in our case the PMPro) optimizes the new frequency request (desired performance), taking into consideration the maximum, lowest and reduction tolerance performance factors. Also, the frequency can be adjusted at the granularity of CPU cores pairs, which are called PMDs, and can range from 300 MHz up to 2.4 GHz at 300 MHz steps in X-Gene 2 and from 375 MHz to 3.0 GHz at 100 MHz steps in X-Gene 3.

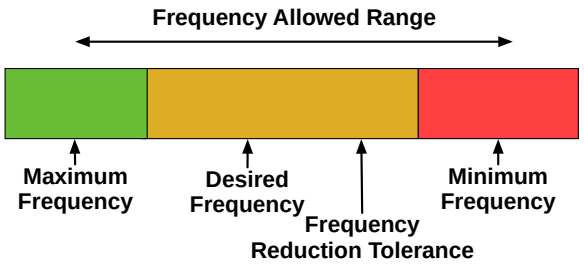


Figure 2.3: CPPC API overview.

Supply voltage adjustment

Moreover, both microprocessor chips have three independently regulated voltage domains the PMD (Processor MoDule), PCP (Processor Complex) and the Standby Power domain, with which we can selectively regulate the voltage independently. The power domain that includes the CPU cores as well as the cache memories hierarchy is the PMD (Processor MoDule) domain (shown in Figure 2.4) and is the one that consumes the largest part of the overall power consumption. All the PMDs (4 PMDs in X-Gene 2 and 16 PMDs in X-Gene 3) operate at the same voltage which can change from the nominal 980mV downwards in X-Gene 2 and from the nominal 870mV in X-Gene 3. The PCP domain includes the operating voltage of the DRAM controllers, the central switch and the I/O bridge. Also, the Power Standby domain includes the SLIMpro and PMpro microcontrollers and the corresponding interfaces for the I^2C buses.

Power and performance monitoring mechanisms

The SLIMpro mechanism can read and report the current power consumption of all power domains of the X-Gene processors. In contrast with Intel CPUs, the X-Gene processors do not feature any power capping mechanism. However, they do offer Performance Monitoring Units which capture the computational characteristics of applications. For both X-Gene processors only up to 4 performance counters per core can be monitored simultaneously.

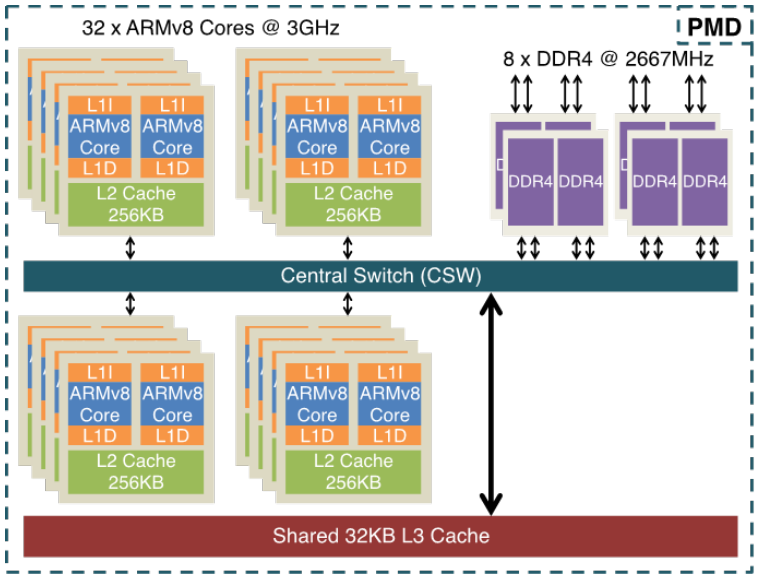


Figure 2.4: Overview of PMD domain of X-Gene processors [12].

2.2 Centaurus Runtime & Programming Model

This Section provides an overview of the Centaurus Runtime & Programming model [13], which we use for the study on the combination of heterogeneous and approximate computing which we discuss in Chapter 7.

2.2.1 Platform model

Centaurus assumes a heterogeneous system, consisting of several accelerator devices capable of performing computations independently of the main host CPU. Each of these devices is typically based on a different hardware architecture, thus contributes a unique set of capabilities to the overall system. Energy consumption may be significantly different between devices too, depending on the architecture, semiconductor fabrication geometry, and device utilization. Also in the general case, the availability of each device on a heterogeneous system is not known at any given time. Devices may be busy or disabled depending on the overall system configuration.

The Centaurus platform inherits the typical OpenCL [14] platform model, depicted in Figure 2.5, providing an abstract execution model for all available devices. That means as a rule, that the programmer can not – in general – make assumptions such as which device executes which computation, however, they can suggest, or even enforce execution on a specific device.

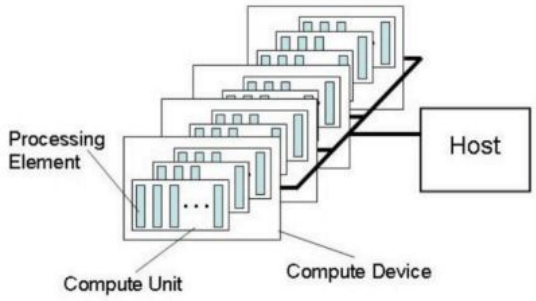


Figure 2.5: OpenCL Platform Model [14]

2.2.2 Execution Model

The execution model also resembles the OpenCL one: the main computation is organized in tasks, each implemented as an OpenCL kernel. The tasks execute on one or more devices, and a host program manages task creation and synchronization.

The programmer defines and spawns tasks from the host program, annotated with information about their significance for the quality of results and data dependencies.

The compiler generates different code versions for each case, accurate and approximate. The underlying Centaurus runtime system is responsible for orchestrating the execution and the selection of the accurate or approximate implementation for each task, based on information provided by programmers/end-users through directive clauses. Figure 2.6 depicts the life cycle of a task, from creation to completion.

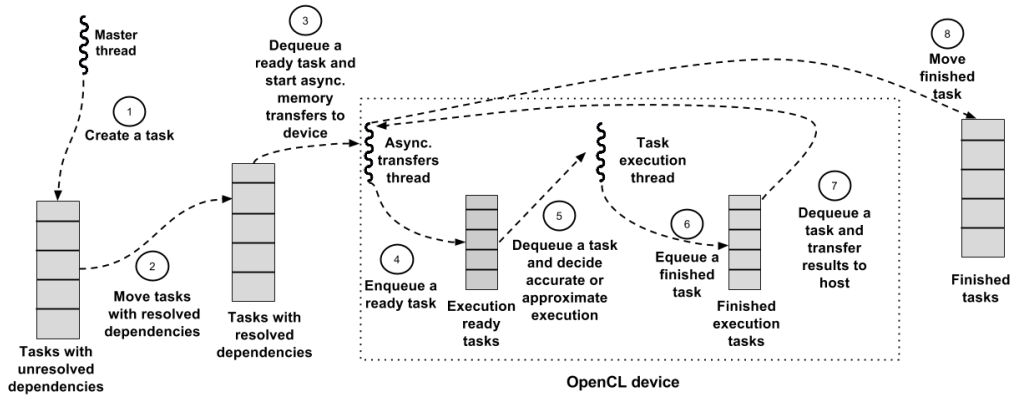


Figure 2.6: Task life in the Centaurus framework.

A newly spawned task usually has to wait on a queue until all its data dependencies are resolved. Then the runtime transfers its input data to the OpenCL device it will be executed on, executes it either accurately or approximately and then transfers back the output results to the host memory.

2.2.3 Directives

Throughout this Section we employ Discrete Cosine Transform (DCT), a code module used in many multimedia algorithms, as a minimal example to illustrate the use of the main programming model concepts. DCT transforms image blocks to blocks of frequency coefficients. Coefficients corresponding to lower spatial frequencies are more significant for the quality of the final image, due to the fact that the human eye is more sensitive to those frequencies. Listing 2.1 outlines the implementation of DCT.

Listing 2.2 summarizes the *#pragma* directive used for task creation. The body of the accurate implementation of the task is defined as a function call, which corresponds to an OpenCL kernel. We explain each clause referring to our DCT example in Listing 2.1.

The *significant()* clause specifies the relative significance of the computation implemented by the task for the quality of the output, with a value (or an expression) in the range [0.0, 1.0]. If set to 1.0 or omitted, the runtime will always execute the task accurately. If set to zero, the runtime will always execute the task approximately, or

```
1  void dctAccurate(double *image,double *result,int subblock) {/*OpenCL code*/}
2  void dctApprox(double *image, double *result, int subblock) {/*OpenCL code*/}
3
4  int subblocks=2*4, subblockSize=4*2, blockSize=32, imgW=1920, imgH=1080;
5  /*DCT block to 2x4 subblocks with different significance, image dimensions*/
6  double sgnf_lut[] = {  1,  .9, .7, .3,
7                         .8,  .4, .3, .1};
8  void DCT(double *image, double *result, double sgnf_ratio) {/* entry point */
9      for (int id = 0; id < subblocks; id++) {  /*spawn dct task group*/
10         #pragma acl task in(image) out(&result[id*subblockSize]) \
11                          label("dct") \
12                          significant(sgnf_lut[id]) approxfun(dctApprox) \
13                          workers(blockSize, blockSize) groups(imgW, imgH)
14         dctAccurate(image, result, id);
15     }
16     #pragma acl taskwait ratio(sgnf_ratio) label("dct") /*execution barrier*/
17 }
```

Listing 2.1: Programming model use case: Discrete Cosine Transform (DCT) on blocks of an image.

even discard it if an approximate implementation is not available. The DCT example defines each task's significance at line 12 using values from a lookup table defined at line 6.

The *approxfun()* clause allows the programmer to provide an alternative, approximate implementation of the task. This is generally simpler and less accurate (may even return a default value in the extreme case), however has a lower energy footprint than its accurate counterpart. For example, *dctApprox()* which is defined at line 12 may set all coefficient values equal to zero. Both accurate and approximate functions take the same arguments which are handled in an abstract way by the runtime system.

The programmer explicitly specifies the input and output arguments of each task with the *in()* and *out()* clauses. Line 10 shows the data dependencies of our DCT example. The corresponding information is exploited by the runtime system for dependence analysis and scheduling, as well as for data management. We also support array ranges, in the form of *array[i:i+size]*, as arguments to *in()* or *out()* clauses to further reduce unnecessary data transfers.

The programmer can explicitly associate a task for execution on a specific device using the *bind()* clause. This limits the flexibility of the programming model, however it proves useful in case an implementation is optimized for a specific device or would not be executed efficiently in certain devices. The *suggest()* clause is a relaxed version of *bind()*; it serves as a hint to the runtime, which can however be ignored. We do not use this feature in Listing 2.1, however a possible usage would look like this: *bind(GPU_ID)*.

To specify the work-items and work-groups geometry for kernel execution the programmer uses the *workers()* and *groups()* clauses, which follow the semantics of

```
1 #pragma acl task [significant( expr )] [approxfun( function )] \
2                  [in( varlist )] [out( varlist )] \
3                  [bind( device_type )] [suggest( device_type )] \
4                  [workers( int_expr_list )] [groups( int_expr_list)] \
5                  [label( "name" )]
```

Listing 2.2: #pragma acl task

```
1  #pragma acl taskwait [label("name")] [ratio( double )]
```

Listing 2.3: #pragma acl taskwait

local- and global work size of OpenCL, respectively. We specify the desired geometry at line 13 as a function of the input image dimensions.

Finally, the *label()* clause can be used to associate tasks with named task groups. Each group is characterized by a unique identifier (group name, given as a string). In this example, line 11 qualifies our task group created at line 10, with the name *"dct"*. The programmer can synchronize and control the quality of computations between tasks within the same group using the taskwait directive which we introduce in Listing 2.3.

The *taskwait* directive specifies an explicit synchronization point, acting as an execution and memory barrier. By default, taskwait waits on all issued tasks so far, unless the *label()* clause (16) is present, which limits the explicit barrier only to tasks of the specific task group. In our example the control flow waits for the *"dct"* task group to finish (line 16).

The *ratio()* clause accepts as an argument a value (or an expression which results to a value) ranging in [0.0, 1.0]. It specifies the minimum percentage of tasks of the specific group that the runtime should execute accurately. *ratio* is a single knob which allows the programmer or the user to control the energy footprint / quality tradeoff. For our DCT example we let the user set the acceptable ratio boundary through the *sgnf_ratio* variable, as shown at line 16.

Furthermore, our DCT example in Listing 2.1 shows: (a) how the programmer can spawn and group together tasks of different significance (lines 10-13), and (b) the usage of an explicit barrier (line 16) with a specified *ratio()*.

Chapter 3

The Impact of CPU Voltage Margins on Power-Constrained Execution

In this Chapter, we focus on the question "How to exploit CPU voltage margins reduction?". More specifically, we study the effects of carefully reducing the aforementioned static, workload-independent part of the CPU voltage guardbands in power capped environments. We focus on CPU power consumption, power consumption at the node-level (at the plug), performance/throughput and CPU temperature. We compare power capping when operating at reduced margins against conventional hardware, software, and hybrid (a combination of hardware & software) approaches under nominal voltage margins. We also investigate whether operation at reduced margins can be combined with conventional power capping approaches.

Our evaluation involves three different off-the-shelf CPUs, namely an Intel x86-64 (Xeon E3-1220v5) and two ARMv8 (AppliedMicro X-Gene 2 and X-Gene 3), which are used to run a mix of benchmarks from various suites, such as SPEC CPU2006 [6], CloudSuite [15], Parsec [16], as well as various stress-tests [17, 18, 19].

Our experimental study shows that reducing CPU voltage margins can significantly improve performance, reduce CPU & node power consumption, as well as CPU temperature. More specifically, reduction of voltage margins results in improved performance compared with conventional power capping approaches by 64%, 30% and 34% on average for Xeon E3, X-Gene 2 and X-Gene 3, respectively. Also, it outperforms hybrid power capping solutions, namely PUPiL [11], by 24%, 22% and 5% on average for Xeon E3, X-Gene 2 and X-Gene 3, respectively.

The main contributions and outcomes of our work are the following:

1. We characterize the voltage margins of the x86-64 Skylake processors for both single- and multi-instance/threaded benchmarks. To the best of our knowledge, we are the first to evaluate the voltage margins of a 14nm (Skylake) processor and also the first to evaluate the V_{min} of x86 multi-core CPUs using

multi-instance/threaded benchmarks. Our offline characterization spans from common benchmark suites to stress tests and micro-viruses.

2. To the best of our knowledge, this is the first work that studies the interplay between CPU power capping and operation at reduced margins concerning various metrics and across multiple platforms.

3. We experimentally, on real systems, demonstrate that conventional CPU power capping mechanisms can be combined with operation at reduced voltage margins.

4. We introduce Reduced Voltage Scaling power Capping (RVSCap), a novel CPU power capping mechanism that operates the CPU at reduced voltage margins, minimizes the performance penalty of power capping at restrictive caps, reduces the power footprint at more relaxed caps and supports multiple platforms.

3.1 Characterization of Voltage Margins

Since we investigate the interplay between reducing CPU voltage margins and power capping mechanisms, it is necessary to quantify the voltage margins of the target platforms through a characterization process. We employ the $(\text{XM})^2$ framework (discussed in Chapter 6) and apply a methodology similar to that in prior work [20, 21, 22, 23, 24], for the CPUs of Table 2.1, using SPEC CPU2006 [6], Parsec [16] and micro-virus [24] benchmarks.

More specifically, for each CPU frequency point we run each benchmark 20 times at each voltage step, in the entire range from the nominal voltage point down to the lowest sub-nominal voltage, referred as V_{min}, at which *any* workload executes successfully without any hardware reported error, silent data corruption (SDC), or system crash. As an additional robustness test for the value of V_{min}, we execute each benchmark 1000 more times and verify that all runs finish successfully. For SDC detection we compare the output of each workload under reduced voltage margins with the corresponding one produced by nominal execution. This V_{min} quantification methodology is conservative, because it misses workload-dependent opportunities to further reduce V_{min}, yet we opt to potentially sacrifice some additional power efficiency to mitigate and preclude any erratic behavior, such as SDCs, when voltage margins are aggressively reduced [21, 22, 24].

Figure 3.1 illustrates the V_{min} identified by our characterization for a range of operating frequencies for all three CPUs. The grey area shows a configuration where —

during our extensive characterization — at least one workload did not execute successfully. The point at the boundary of the green area (safe operation) and grey area (unsafe operation) corresponds to V_{min}. For the Xeon E3 processor, which supports DVFS, we observe that V_{min} drops with frequency reduction. In terms of absolute values, the width of voltage margins ranges from 240mV to 170mV depending on the CPU operating frequency. The observation that the width of voltage margins varies is critical, as the voltage manipulation mechanism of the Xeon E3 processor is offset-based; the different width of voltage margins across different CPU frequencies requires extra care to be taken when transitioning between frequencies.

For the X-Gene 2 and X-Gene 3 processors, which only support dynamic frequency scaling (DFS), the nominal operating voltage is constant for all frequencies, at 980mV and 875mV, respectively. Moreover, major differences in V_{min} can be observed only after reducing the frequency to half of that used by the highest performance operating point. Both X-Gene processors use clock skipping in combination with clock division to generate the effective frequency of processor cores [21]. Whenever the target frequency is less than half of the maximum clock frequency, clock division is first applied. All other intermediate frequency points are implemented with clock skipping, either on the original or on the divided clock frequency. This explains why we do not observe any variation of V_{min} at intermediate frequencies, with relatively large V_{min} differences at frequencies where clock division is applied. Voltage margins are, again, wider at lower frequencies.

The outcome of this characterization (shown in Figure 3.1) is an extended voltage-frequency curve for each CPU, which, for each frequency, provides the sub-nominal voltage V_{min} deemed to be safe by the characterization campaign for all workloads.

We note that the aging effects of the CPUs are out of the scope of this study. Also, we do not investigate system operation at extreme temperatures. Prior research [25, 26] indicates that CPU voltage margins are wider at higher temperatures, due to the transistor-level effect of temperature inversion. However, other components such as disks [27] and DRAMs [28] are more failure-prone at higher temperatures, which makes it difficult to reason about node reliability at extreme temperature conditions. For this reason, in our experiments we operate all our machines under typical datacenter conditions [29] (at an ambient temperature of 21-23 °C), even though this is expected to result in narrower CPU voltage margins.

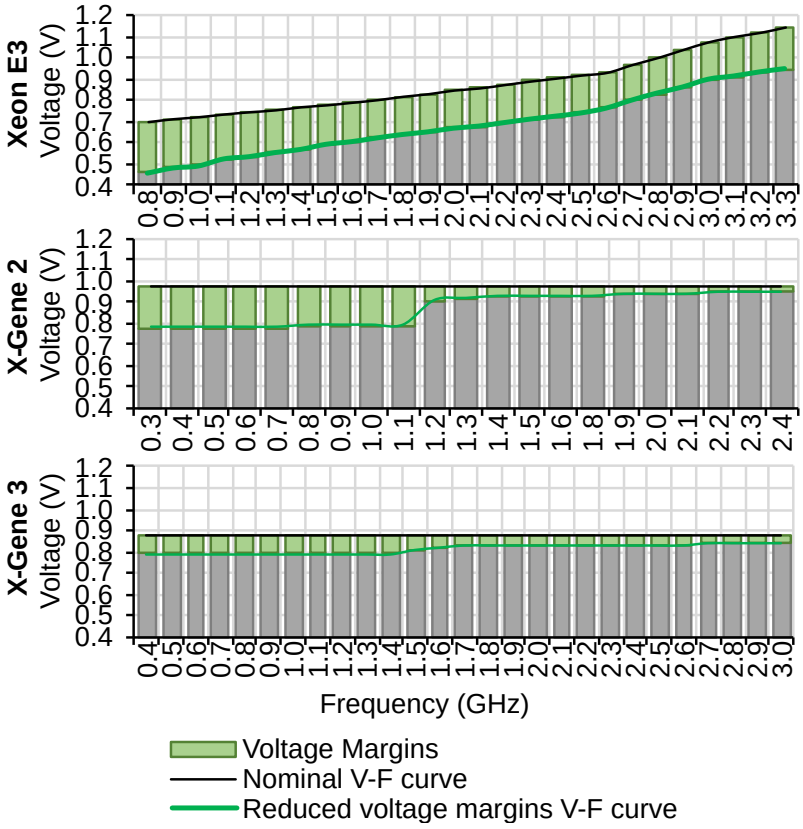


Figure 3.1: Voltage (y-axis) vs. frequency (x-axis) for each CPU.

3.2 Power Capping Approaches

3.2.1 Existing techniques

This Section discusses how the available power capping approaches enforce power caps. There are numerous software-based capping approaches in the literature [30, 31, 32, 33], which use CPU performance counters to monitor key performance metrics. These metrics are then used to calculate the CPU frequency downscaling required to enforce the cap.

Also, modern Intel CPUs, as already mentioned in Section 2.1.1, feature RAPL [10], a hardware power capping mechanism that directly observes the power consumption of the CPU. On the other hand, the X-Gene processors do not provide a hardware capping mechanism.

PUPiL [11] is a recent technique for maximizing the performance of multithreaded applications on Intel x86-64 systems. PUPiL uses RAPL for power capping and manipulates thread allocation to cores based on a heuristic thread-packing algorithm for identifying the best performing threads-to-cores ratio, where the number of threads is larger than the number of cores. PUPiL measures performance indirectly using heartbeats [34] which are generated by the executing workloads.

3.2.2 RAPL-RM

To investigate the impact of exploiting voltage margins on power capped environments, we combine the existing RAPL mechanism with reduced voltage margins (RAPL-RM). Similarly to RAPL, RAPL-RM supports only Intel CPUs. Also, given that RAPL switches between frequencies under hardware control, in RAPL-RM we conservatively apply the safest Voltage offset that corresponds to the narrower voltage margin (170mV) across all frequencies, missing the opportunity to further decrease the voltage by up to 70mV at specific frequencies.

3.2.3 RVSCap

To maximize the power gains introduced by reducing CPU voltage margins and hence minimize the performance penalty induced by a power cap, we design and implement Reduced Voltage Scaling power Capping (RVSCap), a software-based power capping mechanism that reduces CPU voltage by shaving-off pessimistic margins. Also, unlike RAPL and the RAPL-RM extension discussed above, RVSCap is a software-based mechanism that can be employed in widely different platforms, merely by implementing a few platform-specific components.

Similarly to RAPL and RAPL-RM, RVSCap is a closed-loop mechanism that measures the average power consumption of the CPU at the boundaries of time windows, identifies the CPU operating point that satisfies the specified power cap (based on the equation $P = CV^2F$), and then enforces that operating point for the following time window. However, in contrast to RAPL and RAPL-RM, RVSCap includes all the necessary logic mentioned in Section 3.1 for safe transition between different (V_{min},F) pairs. This is critical, as the width of the voltage margin varies across CPU frequencies and a premature adjustment of CPU frequency without properly setting the V_{min} will result in a system crash.

As shown in Figure 3.2, when RVSCap needs to take action because the current power consumption (as reported by the CPU at execution time, e.g. through RAPL on Intel and SLIMPro on X-Genes) is above the defined CPU power threshold, it first checks if the cap can be met by scaling the CPU voltage to a sub-nominal, yet safe level for the current operating frequency, based on the V_{min} quantified via the offline characterization (Section 3.1). If the estimated power reduction by voltage scaling is not sufficient, RVSCap re-calculates the power consumption for the immediately lower operating frequency and the corresponding V_{min}. This is repeated, in an iterative fashion, until an operating point is found that meets the desired power cap, which is then applied to the CPU.

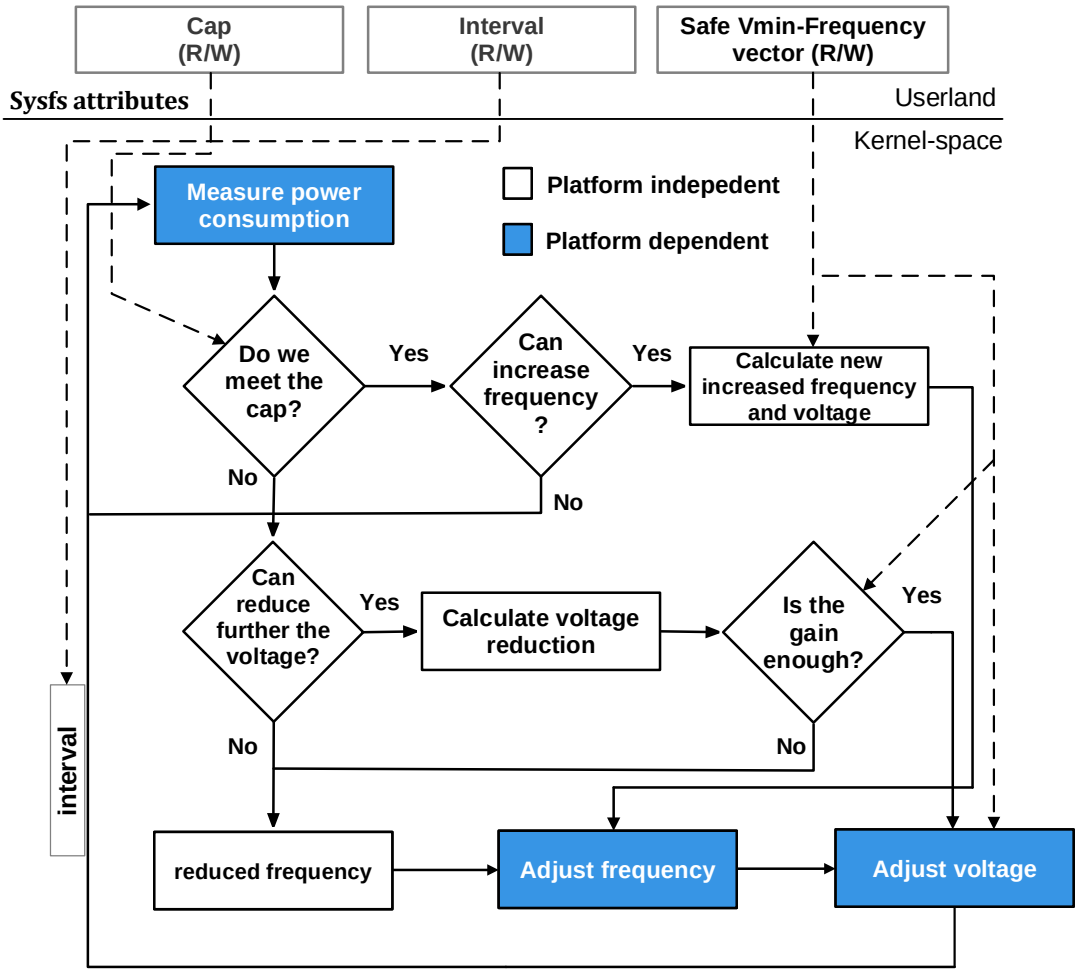


Figure 3.2: Flowchart of RVSCap. Only a few components (in blue) are different and need to be developed for each target platform.

In the opposite case, when the current power consumption is below the threshold, RVSCap calculates and applies the operating point with the highest frequency that meets the power cap. In case RVSCap applies the maximum CPU frequency, the exploitation of voltage margins translates to lower CPU power consumption. This additional power budget slack can be used to power-on additional nodes, thus increasing the active computational capacity of the datacenter or can be exploited towards reducing power and cooling costs.

RVSCap is a simple, lightweight mechanism, that supports multiple platforms and can be used by any user-level runtime system. More specifically, RVSCap can support any platform that exposes the CPU power consumption and allows independent adjustment of the CPU operating frequency and voltage. To port RVSCap to a new platform, one only needs to implement a small number of platform-dependent components that provide the high-level (platform-independent) API used by the rest of the mechanism (see Figure 3.2).

The effective cap enforcement interval depends on the low-level mechanisms that

are available in the underlying platform. We experimentally find that for the Intel Xeon E3 CPU an effective interval of cap enforcement can be as low as 10 msec with 0.29% performance overhead. On the other hand, this interval is much higher, at approximately 1 sec, for the X-Gene platforms. This is because every request of the capping mechanism (reading the CPU power consumption or setting the CPU operating frequency) has to go through an I^2C interface to the SLIMpro processor and then to the PMpro processor.

3.3 Experimental Study

In this Section, we discuss the results of a detailed experimental study of the effects of exploiting reduced voltage margins during power-constrained execution. The study has been conducted using three different platforms that feature the CPUs in Table 2.1. We discuss results in terms of performance, CPU, and node power consumption, as well as CPU operating temperature. More specifically, we capture the CPU power consumption as reported through RAPL for Xeon E3 and SLIMpro for X-Genes. Also, CPU operating temperature is reported through MSRs for Xeon E3 and through SLIMpro for X-Genes. For node power consumption we utilize an external power meter at the plug of each system. Moreover, we combine execution at reduced voltage margins with other state-of-the-art power-capping approaches, such as RAPL and PUPiL and compare against state-of-the-art techniques based on execution at nominal operating points.

3.3.1 Benchmarks

We use 16 benchmarks from SPEC CPU2006 [6], CloudSuite [15], Parsec [16], and individual stress-tests [17, 18, 19] for FP computations, thermal load, and memory use, as shown in Table 3.1. Note that the CloudSuite benchmarks and the stress-tests are not a part of the characterization process, presented in Section 3.1. We include them in our experimental study, as an extra validation of the safety of V_{min} values identified during the characterization process. For parallel and sequential benchmarks we manipulate the degree of parallelism and the number of co-executing instances respectively, to achieve maximum core utilization and maximize the CPU power footprint. CloudSuite benchmarks are the only exception, since they can not be scaled in a practical manner.

To better understand the behavior of the benchmarks of Table 3.1 in a power-constrained execution environment, we categorize them into three groups according

Table 3.1: Benchmark set.

Suite	Benchmark	Ref. name	Throughput metric	Input
Cloud-Suite [15]	Data Analytics	danal	MB/sec	default
	Data Caching	dcach	requests/sec	default
	Web Search	wsrch	searchDriver	default
	Web Serving	wserv	elggDriver	default
Parsec [16]	Blackscholes	blcks	prices/sec	native
	Facesim	fcsim	frames/sec	native
	Fluidanimate	fanim	steps/sec	native
	Swaptions	swpts	swapt./sec	native
SPEC CPU 2006 [6]	Bzip2	bzip2	MB/sec	ref
	Bwaves	bwves	steps/sec	ref
	Gromacs	grmcs	frames/sec	ref
	h264Ref	h264r	frames/sec	ref
Stress-tests	Memstress	mstre	Bytes/sec	default
	Linpack [17]	lpack	equat./sec	default
	Prime95 [18]	mprime	primes/sec	default
	Firestarter [19]	fires	fma ops/sec	default

to their instruction retirement rate, thus, implicitly according to their compute intensity. Figure 3.3 shows the percentage of retired micro-operations (uops), averaged across all 4 CPU cores of the Xeon E3, for each benchmark. Following Intel's Top-Down Microarchitecture Analysis Method (TMAM) [35], the remaining uops are attributed to bad-speculation, as well as to back-end and front-end bound operations, which cover several types of stalls that prevent the delivered uops from retiring. Similarly, Figure 3.4 depicts the retired instructions per cycle (IPC) averaged across all 8 CPU cores of the X-Gene 2, for each benchmark. Moreover, the behavior is very similar, as can been seen by comparing Figures 3.4 and 3.5, on the 32-core X-Gene 3 as well.

Most benchmarks in CloudSuite have low compute intensity, as they do not utilize all the available CPU cores. Interestingly, some benchmarks exhibit different behavior on the Xeon E3 and X-Gene 3. For example, *facesim* is more compute-intensive than *Blackscholes* on the Xeon E3, whereas it turns out to be (far) less compute-intensive on the X-Gene 3, where its execution leads to a larger number of slow memory accesses. This is due to the higher thread contention for capacity at the last level cache of X-Gene. On Xeon E3, 4 threads share 8MB of L3, whereas on X-Gene 3 all 32 threads share 32 MB of L3.

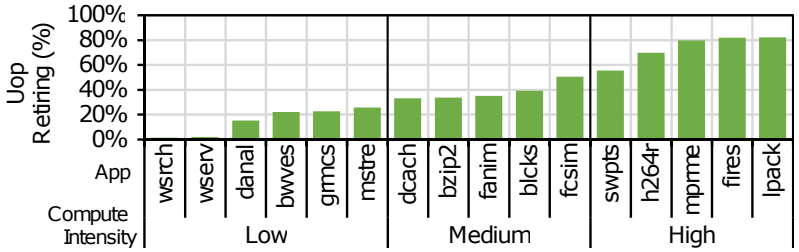


Figure 3.3: Uops retiring(%) normalized wrt. CPU max performance (Table 2.1) on Xeon E3.

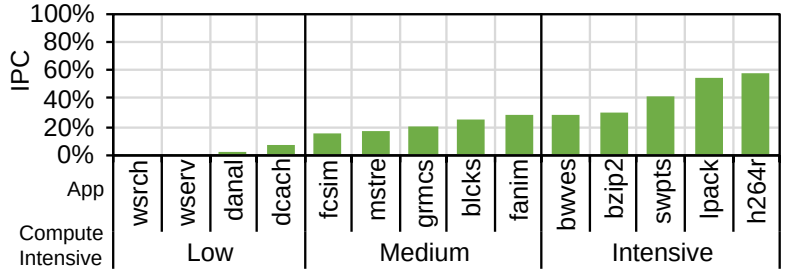


Figure 3.4: Retired instructions per cycle (IPC)(%) normalized wrt. CPU max performance (Table 2.1) on X-Gene 2.

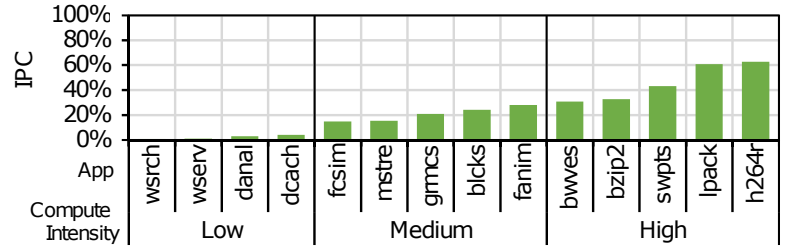


Figure 3.5: Retired instructions per cycle (IPC)(%) normalized wrt. CPU max performance (Table 2.1) on X-Gene 3.

3.3.2 Effects of CPU voltage margins on power capping

In this Section, we quantify the impact of exploiting reduced CPU voltage margins in power capped environments, for various CPU power capping mechanisms.

Figure 3.6 reports the power consumption of the Intel Xeon E3 processor platform for an aggressive, a relaxed, and a conservative power cap when executing Data-caching, Fluidanimate, and Bzip2 applications. To this end, as aggressive, we set the lower power threshold that can be achieved only by scaling the CPU operating frequency, namely 30W and 10W for X-Gene 3 and X-Gene 2, respectively. As conservative, we set a power cap which can be achieved with no frequency scaling for all the workloads in our evaluation. The conservative power cap corresponds to 60W for Xeon E3, 80W for X-Gene 3, and 30W for X-Gene 2. Finally, the relaxed cap lies in the middle of the aggressive and conservative caps, namely at 50W and 20W for X-Gene 3 and X-Gene2 respectively.

Firstly, we observe that RVSCap strictly adheres to the given power constraints,

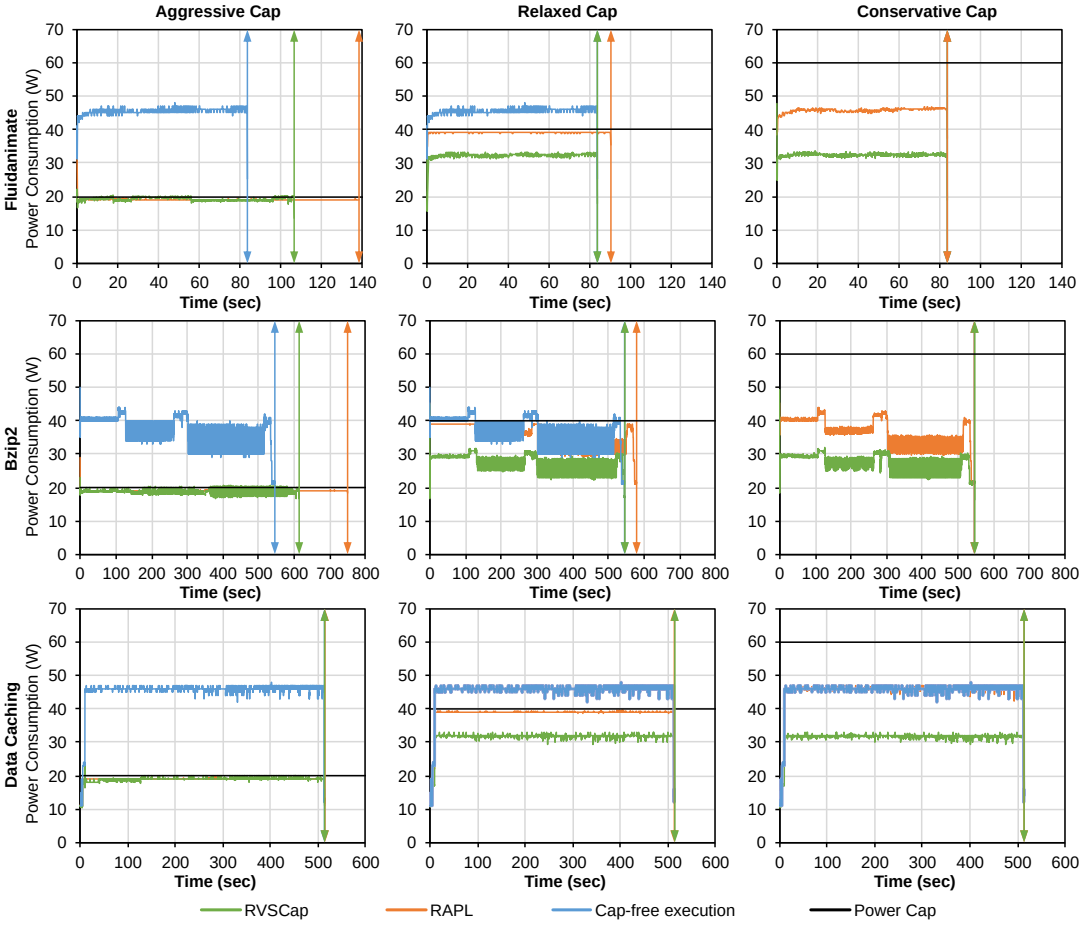


Figure 3.6: Power consumption on Skylake for different applications (rows), power caps (columns) and power capping methodologies (lines in each chart). The blue line corresponds to cap-free execution in nominal settings. Vertical lines with double arrows indicate the completion time of each execution.

thus can provide the fundamental functionality of a capping mechanism. The distance between the blue and orange vertical double arrow lines, which mark the execution completion time of each benchmark without any capping (cap-free execution) and the conventional capping mechanism (RAPL) respectively, indicates the performance penalty due to conventional power capping. Similarly, the difference between the green and orange vertical double arrow lines, the latter marking the completion time of the benchmark with RVSCap, indicates the performance gains of RVSCap compared with the conventional power capping mechanism. Unlike the other two benchmarks, *Data Caching* runs for a fixed amount of time. In this case, the throughput metric is requests per second, thus the performance impact due to power capping is not visible in the plots. For the *aggressive* power cap of 20 Watts, RVSCap delivers 22% higher throughput for *Fluidanimate* and *Data Caching*, and 18% for *Bzip2*, compared to RAPL. For higher power budgets, RAPL can afford high CPU operating

frequencies and does not suffer any significant performance degradation.

Note that the blue and orange lines completely coincide (blue line is not visible) at the *conservative* cap of 60 Watts in Figure 3.6. In this case, the power cap is higher than the power footprint of the application when executing without any power constraints, so the cap is not restrictive. In such scenarios, RVSCap delivers the same throughput with a smaller power consumption footprint. This is evident for all three benchmarks, when the CPU operates at the maximum frequency. In this case RVSCap achieves 29% less power consumption for Fluidanimate, 30% for *Data Caching* and 26% for *Bzip2*.

Furthermore, for the Xeon E3 CPU, we evaluate the following power capping techniques: i) RVSCap, ii) conventional RAPL, and iii) *RAPL-RM*. Figure 3.7 shows the results. The x-axis of each row represents the benchmarks under different CPU power caps, sorted according to their compute intensity (section 3.3.1). The left y-axis (and the corresponding bars) quantifies the absolute values of each metric. The two lines refer to the right y-axis and show the average per cap relative increase or reduction among all benchmarks of the respective metric, attained by RAPL-RM and RVSCap over RAPL.

We observe that RVSCap outperforms both RAPL and RAPL-RM and depending on the enforced cap the gains translate to improved performance or lower power consumption both at CPU and node level, as well as lower CPU operating temperature. In general, the trend is that for aggressive power caps (15W - 18% of Xeon E3 TDP)

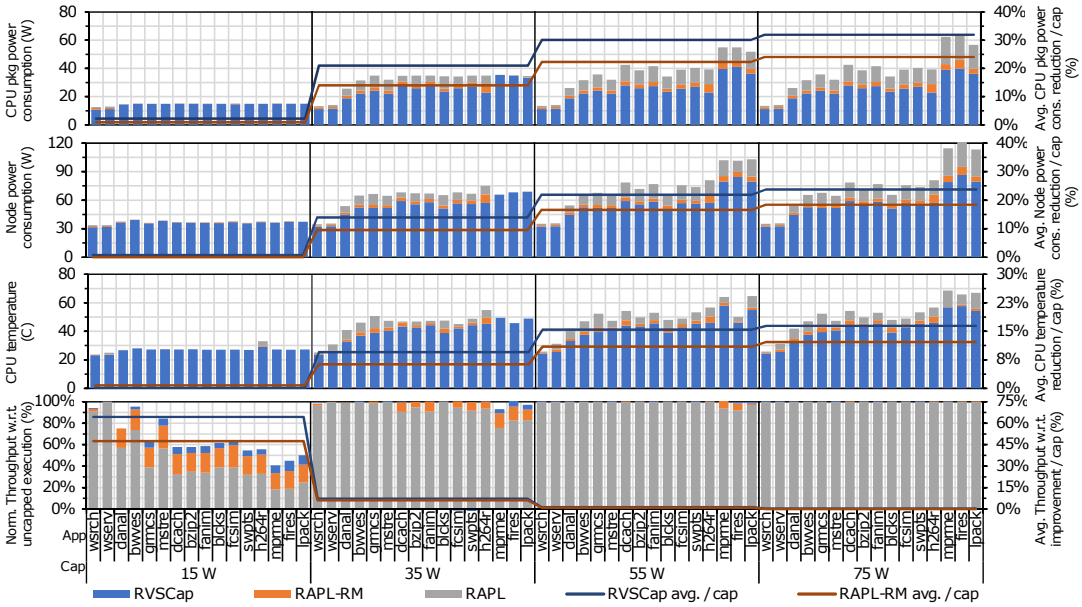


Figure 3.7: CPU and total node power dissipation, CPU temperature and performance for different power caps and power capping mechanisms on Xeon E3. Bar plots correspond to the left y-axis and lines to the right y-axis.

the reduced voltage margins enable significantly improved performance. For higher caps (35W,55W - 43%,68% of Xeon E3 TDP), which are restrictive for fewer benchmarks, the voltage margins reduction results in both performance gains and reduction in power consumption and operating temperature of the CPU. Eventually, for higher caps (75W - 94% of Xeon E3 TDP) where conventional power capping typically does not result in performance penalties, voltage margins reduction results in lower power dissipation and CPU temperatures.

As an example, RVSCap results in a 104% higher performance for *Linpack* under aggressive power caps, while for relaxed power caps it reduces node power consumption and CPU operating temperature by 31% and 19% respectively, compared with RAPL. On average, depending on the enforced cap, RVSCap delivers up to 64% improved performance, 32% lower CPU power consumption, 24% lower power consumption at the node level and 16% lower CPU temperature, compared with RAPL. In comparison with RAPL-RM, RVSCap delivers on average up to 17% improved performance, 8% lower CPU power consumption, 6% lower node power consumption and 4% lower CPU operating temperature. These gains are because RVSCap can dynamically apply the appropriate V_{min} for each CPU operating frequency, rather than using the safest V_{min} across all frequencies.

More specifically, RAPL-RM switches between frequencies under hardware control and lacks the necessary logic for adjusting voltage reduction accordingly, to avoid imminent system crashes, during the frequency transitioning. Consequently, the only safe voltage reduction for RAPL-RM is the one that corresponds to the narrower voltage margin across all CPU frequencies. On the contrary, RVSCap is controls, at the same time, both the CPU frequency and voltage operating points and, hence, can apply the appropriate V_{min} for each CPU operating frequency and achieve higher benefits from voltage margins reduction. Moreover, RVSCap can achieve a 55W power cap without performance throttling on *any* of the benchmarks. On the other hand, in order to enforce the same cap RAPL and RAPL-RM have to throttle the performance in 3 and 2 of the 16 benchmarks, respectively.

Given that X-Gene processors do not support DVFS and do not offer any power capping mechanism, to present a comprehensive comparison with execution at reduced voltage margins enabled by RVSCap, we implement a software power capping mechanism that employs only frequency scaling to emulate a conventional frequency scaling power capping mechanism (DFSCap). More specifically, DFSCap has the same logic as RVSCap, however, it does not consider operating at reduced voltage margins points.

Figure 3.8 presents the power profiles of our experiments on the X-Gene 3 platform. Note that power dissipation fluctuates around the limit for both RVSCap and

conventional power capping. This is because SLIMpro updates the reported power consumption information every 1 second. This architectural limitation restricts RVSCap to just one monitoring / adaptation cycle per second (in the Xeon CPU, RVSCap performs one such cycle every 10 msec). In some cases, this may allow instantaneous power spikes to occur. Still, on average RVSCap conforms to the requested power consumption cap. Figure 3.9 depicts the corresponding results for X-Gene 2, which exhibits the same behavior as X-Gene 3. Also, it should be noted that this long monitoring/adaptation interval, which is a concern on X-Gene 2 and X-Gene 3 processors, is still equal or shorter than intervals used in prior software-based or hybrid software/hardware works [11, 30].

Figures 3.10 and 3.11 illustrate the performance of those mechanisms on the X-Gene 2 and X-Gene 3 CPUs, respectively. The trends are generally similar to what was observed on the Intel-based system. The requested power cap acts as a knob that

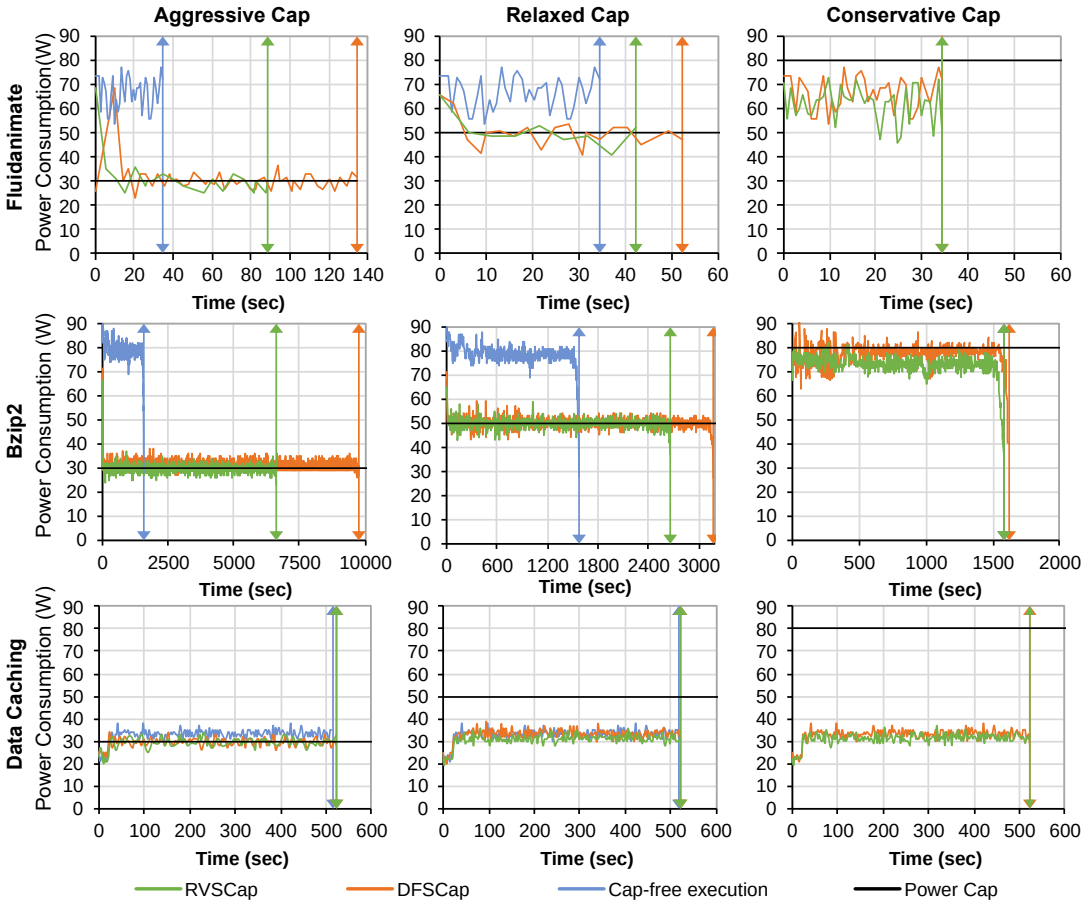


Figure 3.8: Power consumption on X-Gene 3 for different applications (rows), power caps (columns) and power capping methodologies (lines in each chart). The blue line corresponds to cap-free execution in nominal settings. Vertical lines with double arrows indicate the completion time of each execution.

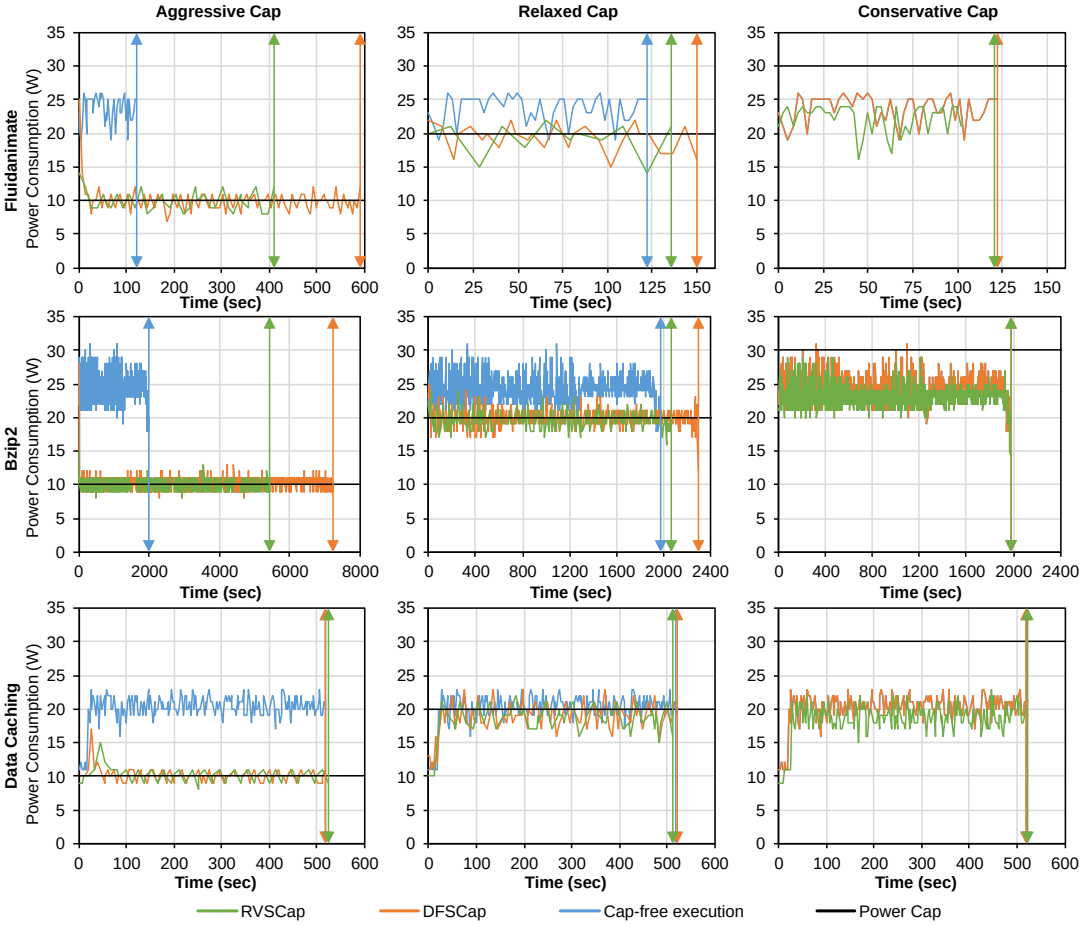


Figure 3.9: Power consumption on X-Gene 2 for different applications (rows), power caps (columns) and power capping methodologies (lines in each chart). The blue line corresponds to cap-free execution in nominal settings. Vertical lines with double arrows indicate the completion time of each execution.

directs the benefits introduced by the reduction of voltage margins towards either performance gains for aggressive caps, or reduced power consumption for higher, less restrictive power caps. Notably, the voltage margins reduction on X-Gene processors do not have a significant impact on CPU operating temperature, as was the case on Xeon E3. This is because the CPU fans on both X-Gene systems operate in two fixed, distinct modes (low and high rate), depending on CPU temperature. On the contrary, for the Intel platform, the rotation rate (rpm) of the CPU fans scales accordingly with the CPU temperature.

On average, depending on the requested cap, RVSCap delivers up to 30% improved performance, 11% reduced CPU power consumption, 4% reduced node power consumption and 1% reduced CPU temperature for X-Gene 2 compared with DFSCap. On X-Gene 3, the respective gains are 34% improved performance, 8% lower CPU power consumption, 5% lower node power consumption and 1% reduced CPU

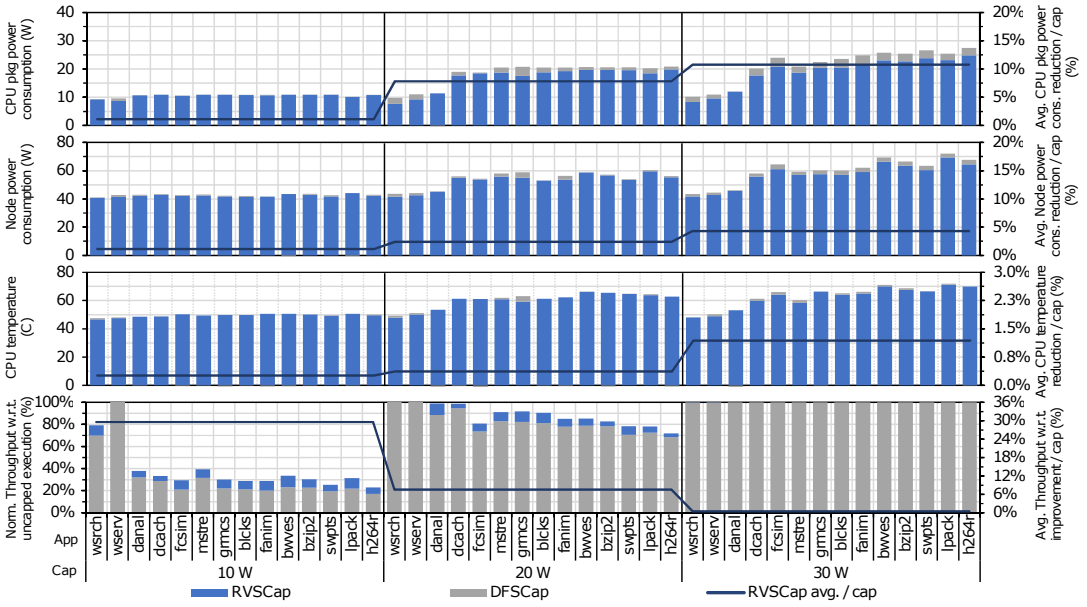


Figure 3.10: CPU and total node power dissipation, CPU temperature and performance for different power caps and power capping mechanisms on X-Gene 2. Bar plots correspond to the left y-axis and lines to the right y-axis.

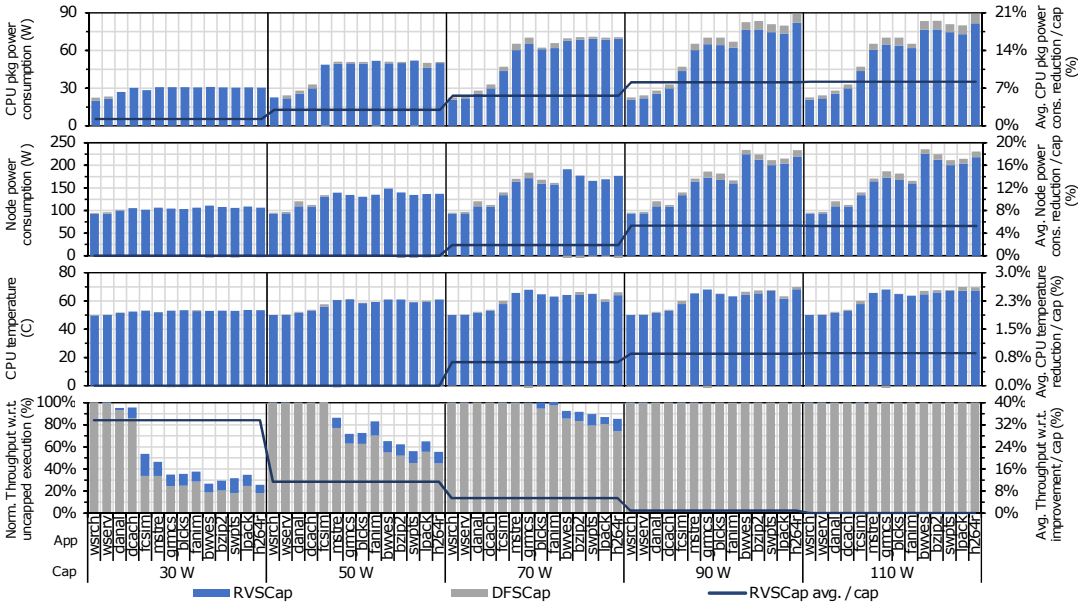


Figure 3.11: CPU and total node power dissipation, CPU temperature and performance for different power caps and power capping mechanisms on X-Gene 3. Bar plots correspond to the left y-axis and lines to the right y-axis.

temperature.

So far, our investigation indicates that the exploitation of CPU voltage margins in a power capped environment (RVSCap), outperforms CPU capping mechanisms that control the DVFS operating points of the CPU, such as RAPL, and the CPU operating

frequency like DFSCap. Table 3.2 summarizes the gains attained by RVSCap over RAPL for Xeon E3 and over DFSCap for X-Gene processors, averaged over the three compute intensity groups of benchmarks, under aggressive and relaxed power caps. We observe that on Xeon E3 the high-intensity benchmark group demonstrates the highest performance benefits for aggressive caps and the lowest power consumption and CPU operating temperature for relaxed caps. For the X-Gene processors we observe that, under aggressive power caps, the highest compute intensity group again has the highest performance gains. Such a behavior is expected, since benchmarks, in the high compute-intensity group, devote most of their execution time to performing computations, in contrast with lower intensity groups which are memory- or I/O-bound. Consequently, the frequency throttling caused by an aggressive power cap directly affects their performance and even the slightest frequency increase will result in a performance increase.

Moreover, modern Intel processors, like the Xeon E3 used in the context of this dissertation, exploit a hardware mechanism, namely *C-States*, which dynamically powers on or off different modules of the CPU, according to the resource requirements of the workload. Due to this mechanism, we observe that, for conservative power caps, the reduction of voltage margins, for the Xeon-E3, results in different energy gains for different intensity groups. However, the X-Gene processors do not feature a similar mechanism and, consequently, the reduction of voltage margins results in similar energy gains irrespective of the compute-intensity of the benchmark.

3.3.3 Combining RVSCap with hybrid power capping mechanisms

As discussed in Section 3.2, hybrid power capping approaches exploit the nominal DVFS points of the CPU (by directly controlling frequency, such as Intel's RAPL

Table 3.2: Average gains of RVSCap, compared with RAPL and DFSCap for Xeon and X-Gene processors respectively, under aggressive and relaxed power caps, for the different compute-intensity groups of the benchmarks.

Power Cap	**Metric**	**Xeon E3**			**X-Gene 2**			**X-Gene 3**		
		Low	*Medium*	*High*	*Low*	*Medium*	*High*	*Low*	*Medium*	*High*
aggressive	Performance improvement	30%	69%	102%	11%	36%	37%	3%	43%	49%
relaxed	CPU power reduction	26%	33%	37%	11%	12%	10%	9%	7%	8%
relaxed	Node power reduction	17%	25%	31%	3%	5%	5%	5%	5%	6%
relaxed	CPU temperature reduction	17%	16%	17%	1%	2%	2%	1%	2%	2%

does) in combination with the placement of threads to cores. PUPiL [11], a state-of-the-art work in CPU power capping, combines RAPL with thread-packing to maximize performance in power constrained executions. In this Section, we evaluate the effects of combining RVSCap with PUPiL to evaluate whether RVSCap is orthogonal to, and can be combined with any mechanism that controls different execution knobs, apart from CPU frequency and voltage, like PUPiL does for CPU cores utilization. On the X-Gene processors, we combine thread-packing with DFSCap and with RVSCap. On the Intel CPU, we combine thread-packing with RAPL (as in [11]) and RVSCap.

Figure 3.12 shows the performance and node power consumption for RAPL, RVSCap, RAPL+PUPiL, and RVSCap+PUPiL. Given that PUPiL [11] is effective on multithreaded workloads only (where thread-packing can be applied), we use three additional multithreaded benchmarks: *k-means*, *jacobi*, and *stream*, which were also not included in the characterization process of the CPUs presented in Section 3.1. To

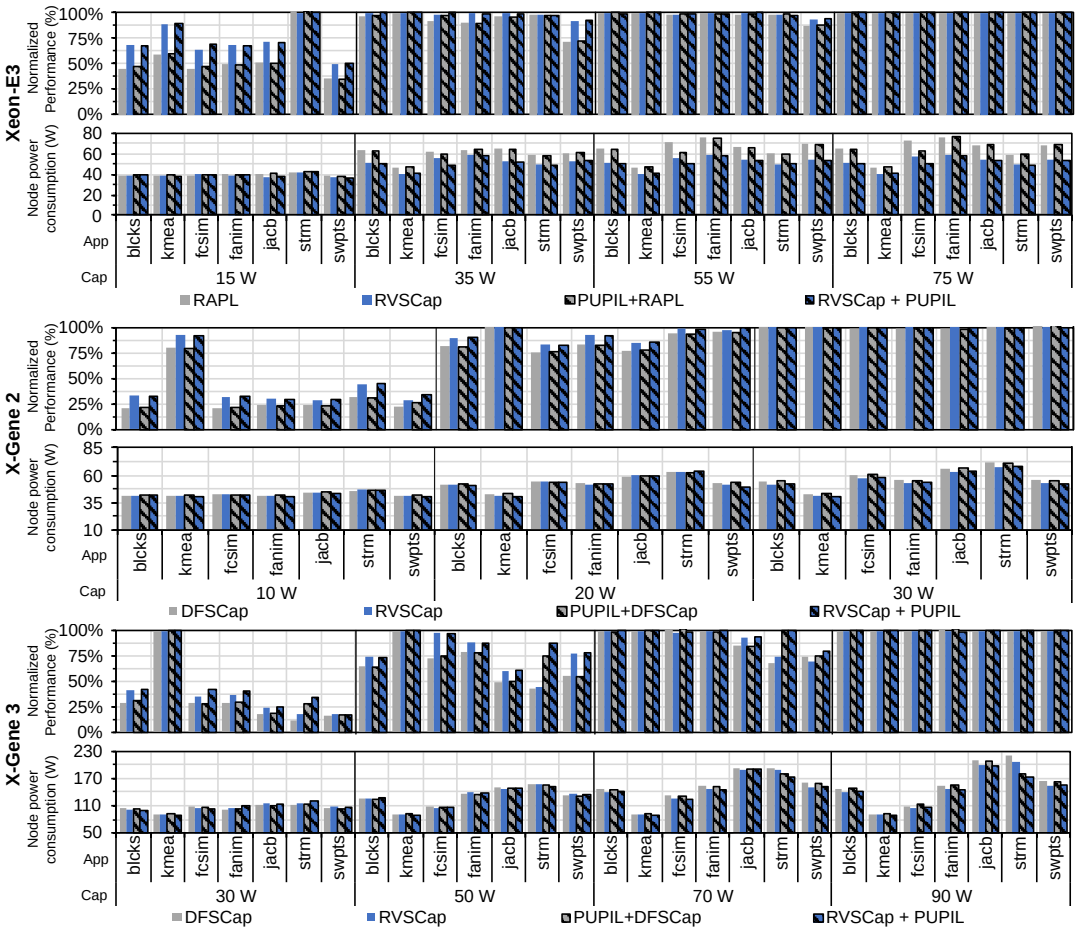


Figure 3.12: Normalized performance (wrt. to uncapped execution) and node power consumption for RAPL, RVSCap, PUPiL, and RVSCap+PUPiL on Xeon E3 and for DFSCap, RVSCap, DSFCap+PUPiL and RVSCap+PUPiL on X-Gene processors.

facilitate a more direct comparison with [11], we measure the performance as the rate of heartbeats that are delivered within a time window (*PUPiL employs a heartbeat framework implemented at the code level of the workload [34]).*

On Xeon E3, RAPL+PUPiL roughly matches RAPL in terms of performance, except for *facesim* where it outperforms RAPL. For restrictive power caps, both RVSCap and RVSCap+PUPiL outperform RAPL+PUPiL on average by 24% and 36%, respectively. This clearly shows that reduced voltage margins can provide additional benefits on top of state-of-the-art hybrid power capping methods. The limited effectiveness of RAPL+PUPiL can be attributed to the fact that Xeon E3 is a single socket system and has only 4 CPU cores, whereas [11] evaluates RAPL+PUPiL on a dual-socket system with a total of 32 CPU cores.

The results are similar for X-Gene 2. The small number of CPU cores limits the performance gains of DFSCap+PUPiL, which is outperformed by RVSCap and RVSCap+PUPiL on average by 22% and 33%, respectively, for restrictive power caps. However, on X-Gene 3, the high CPU core count allows DFSCap+PUPiL to achieve greater performance gains for several benchmarks (such as *facesim* and *stream*). Still, RVSCap and RVSCap+PUPiL continue to outperform DFSCap+PUPiL by 5% and 27% on average, respectively. The only case where DFSCap+PUPiL outperforms RVSCap is *stream* , but the combined RVSCap+PUPiL is the most efficient approach as it takes advantage of both thread-packing and the operation of the CPU at reduced voltage margins.

Our investigation shows that, as expected, thread-packing requires a high CPU cores count to achieve notable performance gains. In contrast, the reduction of voltage margins improves both performance and power efficiency irrespectively of the number of CPU cores. Also, the synergistic operation at reduced voltage margins with thread-packing (RVSCap + PUPiL) is not only possible but also leads to better results than each of these mechanisms separately. Consequently, RVSCap can be used as a drop-in replacement of other power capping mechanisms in the context of more complex, multi-level power efficiency optimization techniques.

3.4 Platforms Comparison

Based on these results, it is evident that the educated reduction of conservative CPU voltage margins significantly reduces the performance penalty introduced by conventional power capping approaches. Moreover, when the power consumption footprint of the workload remains below the power cap, the same performance is achieved with less power consumption.

Recent works [3, 36, 37, 38, 39, 40] try to model and simulate different aspects of datacenters. They focus on maximizing the energy-efficiency and minimizing the costs for operators. Motivated by this line of research, we perform a comparison among the processors at hand, in terms of performance/throughput, to highlight the impact of voltage margin reduction on those aspects of datacenters. Also, since power-constrained execution can result in CPU frequency throttling, we provide a model that predicts the performance degradation of a workload according to the CPU operating frequency.

Figure 3.13 quantifies the percentage of benchmarks for which each CPU proved to be the most efficient in terms of throughput under a specific power cap. We exclude X-Gene 2 from comparisons for power caps higher than the 30 Watt, as those caps exceed its TDP, therefore the comparison would be unfair.

For the conventional power capping mechanisms such as RAPL and DFSCap, as shown in Figure 3.13, Intel Xeon E3 is the most efficient processor, up to the 40 Watt power cap. For higher power envelopes, X-Gene 3 dominates, in terms of throughput. However, when reducing CPU voltage margins with the utilization of RVSCap (see Figure 3.13) Intel Xeon E3 becomes significantly more efficient, approaching the performance of X-Gene 3 for higher power consumption budgets. This is because Intel Xeon E3 has wider voltage margins than X-Gene 3, which results in greater performance improvement under power-constrained execution.

In terms of which is the most appropriate platform for each scenario, despite the narrower voltage margins of X-Gene 2, when the power budget is limited and lies

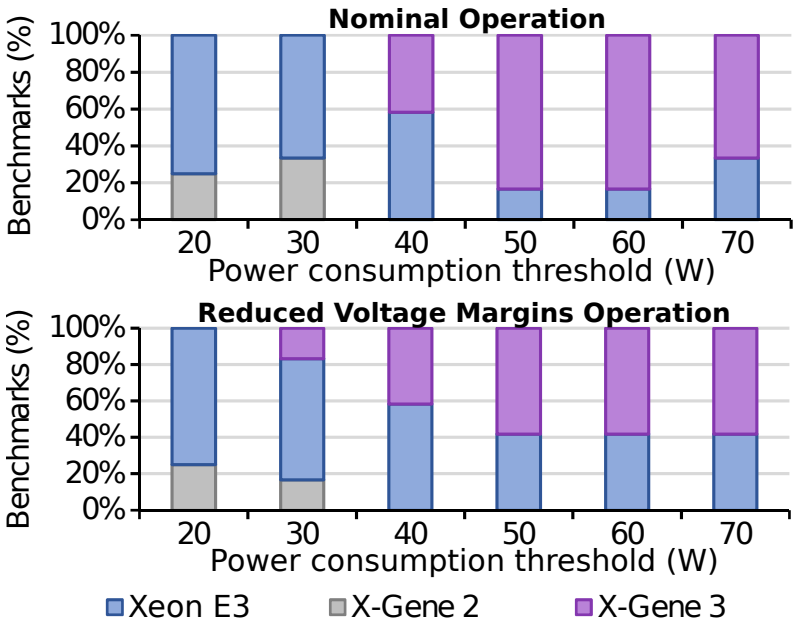


Figure 3.13: Percentage of experiments (benchmarks) for which each processor achieved the highest throughput at a given power cap.

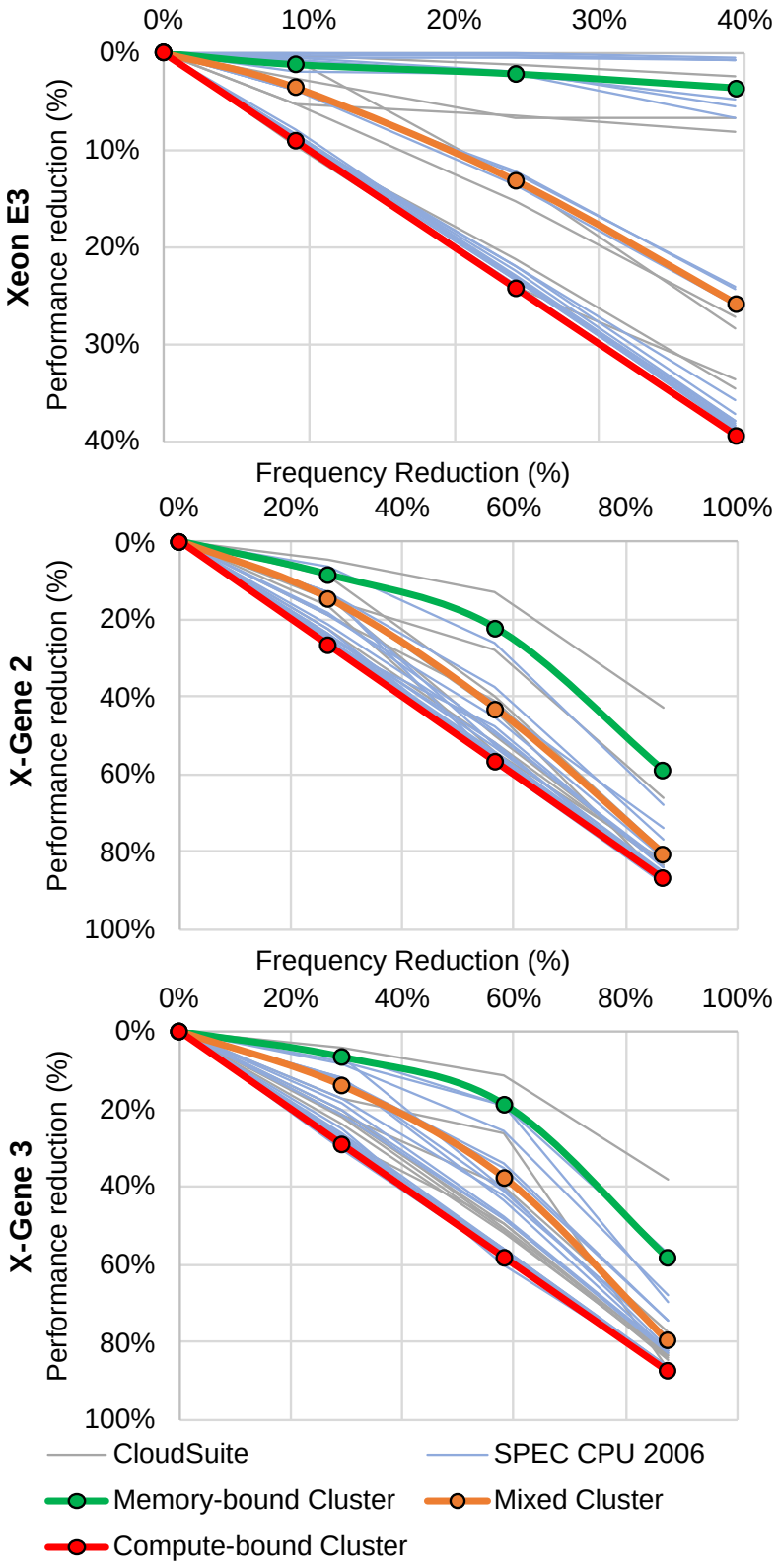


Figure 3.14: Classes of applications, in terms of performance sensitivity to frequency reduction.

below 20 Watts, as is often, for example, the case in Edge computing deployments, this is the only processor that is capable of operating within the designated budget.

For higher power budgets, Intel Xeon E3 delivers sufficient throughput for cloud and single-threaded benchmarks. However, for benchmarks that are inherently parallel, X-Gene 3 is the most efficient CPU due to its higher core count.

The performance of an application depends on CPU operating frequency, but some applications are more sensitive to frequency scaling. To quantify this effect, motivated by our observations in Section 3.3.1, we measure the execution time of the applications, presented in Table 3.1, for each of the frequency points applicable on the Xeon E3, X-Gene 2 and X-Gene 3 processors. After extracting the performance profile for each application on each frequency point, we apply a K-Means clustering algorithm to form clusters of applications with similar behavior.

We identify three main clusters of applications with distinctly different behavior. As shown in Figure 3.14, the applications within each of these clusters exhibit similar performance degradation as a function of frequency scaling on all three platforms. The first cluster, to which we refer as *compute-bound*, includes applications with performance degradation which scales almost linearly to CPU frequency reduction, with a ratio of approximately $1x$. The second cluster, referred to as *mixed*, is less affected by the frequency reduction, however, aggressive frequency reduction leads to significant performance. The third *memory-bound* cluster includes applications that are relatively insensitive to frequency reduction. Moreover, we observe some differentiation, in terms of performance degradation with frequency underscaling, of the aforementioned application clusters between the Xeon E3 and the X-Gene processors. Such differences can be attributed to the completely different architectures of the platforms, different CPU cores count, different size of the processor caches, different RAM size, as well as, different types of system disks (Solid State Drives versus conventional Hard Disk Drives).

To this end, a future line of research could try to exploit modeling the performance of applications, in order to design quality of service- (QoS) aware power capping policies.

3.5 Power Modeling to Mitigate Hardware Limitations

Based on the results of our investigation, we observe that the effectiveness of a power capping mechanism heavily depends on the interval at which the underlying hardware mechanism, such as RAPL for Intel and SLIMpro for X-Gene processors, can report the current CPU power consumption and can apply the CPU configuration parameters instructed by the mechanism. These are the reasons why RVSCap, may momentarily slightly exceed the enforced power cap in the case of the X-Gene processors. To mitigate this effect, in conjunction with the fact that this is a hardware limitation, we

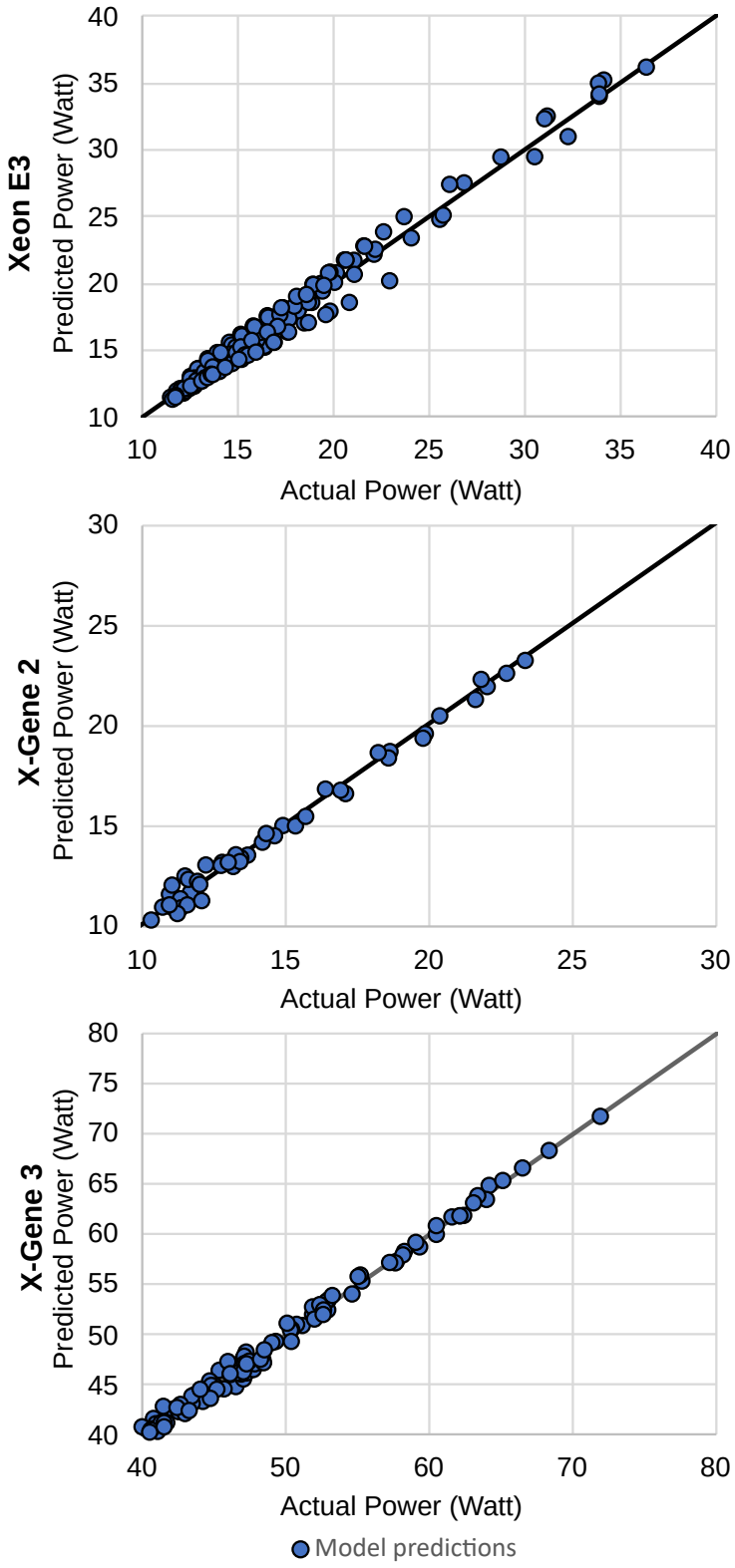


Figure 3.15: Predictions of our power estimation models for Xeon E3, and X-Gene processors.

develop a model that estimates power consumption. Given the model, a power capping mechanism can utilize it to – at least – mitigate the latency of periodic power

monitoring queries. For example, on X-Gene processors RVSCap could maintain a higher interval rate of cap enforcement by employing the power consumption models, for a fixed number of consecutive frames, and, thus, predict the current power consumption until the underlying hardware mechanism provides the actual measurement of the power consumption.

We estimate the power consumption function, as follows: First, we run a series of experiments used, and numerous nominal and underscaled voltage-frequency points. We record the power consumption, measured through RAPL and SLIMPro, for Xeon E3 and X-Gene processors, respectively. Then, based on this data, we use linear regression, using 80% and 20% of the data as training and validation set respectively, to compute the parameters of the model $Power = A + B * u * V^2 * f$ which estimates power consumption as a function of CPU utilization u, supply voltage V and frequency f. This model can then serve as the power estimation function for power capping policies.

As shown in Figure 3.15, for the Xeon E3 CPU, we get $A = 3.03$ and $B = 10.74$, which predicts power consumption with $R^2 = 0.99$ and a root mean square error (RMSE) of 0.78 for the unseen configurations at nominal operating points. For the X-Gene 3 CPU, the values of the parameters are $A = 0.72$ and $B = 17.88$, yielding $R^2 = 0.99$ and an RMSE of 0.77. Similarly, for the X-Gene 2 CPU, $A = 0.84$ and $B = 7.71$, yielding $R^2 = 0.99$ and an RMSE of 0.16. The low RMSE indicates that the predictions of both models are close to the actual power consumption. Furthermore, as an extra validation step, we repeated the series of experiments for configurations at extended margins and we verified that the accuracy of the power estimation models remains in the aforementioned RMSEs.

3.6 Related Work

Several approaches relax voltage guardbands to increase energy efficiency. In particular, [41, 42] present heuristics that dynamically reduce voltage margins, based on the feedback of error correction ECC mechanism built-in Itanium 9560, and [43] rely on dedicated hardware protection mechanisms introduced at chip design-time to reduce voltage margins. Also, [21, 22, 24] authors present an automated system-level analysis on multi-core CPUs based on the ARMv8 64-bit architecture when pushed to operate in scaled voltage conditions. [23] targets the same ARMv8 architecture and combines the voltage margins of both CPU and DRAM modules to improve the potential energy savings. [44] presents, similar to this work, static offline analysis that quantifies the CPU voltage margins on x86-64 processors.

In contrast with our work, presented in this Chapter, all of the aforementioned prior approaches are completely agnostic of CPU power capping, target CPU architectures that do not support hardware power capping, focus on a limited range of CPU frequencies and rely on specific CPU design, topology and behavior to carefully exploit the reduction of voltage margins. Consequently, all these attributes make prior work incompatible with existing power capping approaches. Furthermore, none of the previous research efforts projects the potential energy gains in large-scale deployments, as this work does, where CPU power capping is typically useful, however, a fault tolerance mechanism, such as checkpointing, is also necessary.

Earlier power capping solutions [30, 31] are software-based, rely on external power meters and control CPU key factors such as frequency. However, several research efforts note that coordinating multiple components provides greater performance under a power cap than management of a single component in isolation. More specifically, work [45] focuses on a class of services with very high CPU and memory demands, best represented by internet search, and propose a power-aware resource allocation algorithm for the CPU and the memory which is driven by SLA and allows for various dynamic cluster configurations, from energy-optimal to resource-usage-optimal. Also, the authors in [46, 47] introduce several approaches to multi-component power management, that continue to deliver maximum performance for a given power budget even as new components become available or existing components are disabled. Furthermore, the authors in [48] propose a control-theoretical methodology to complement architecture design and show that their approach meets performance requirements, while consuming less power than any fixed one, and it is capable of attaining the same goals. The authors of [49] present a fine-grain characterization, expose the opportunity for power savings using low-power modes of each primary server component, and introduce and validate a performance model to evaluate the impact of processor- and memory-based low-power modes on the search latency distribution. Also, [50] explores how to integrate power management mechanisms and policies with the virtualization technologies being actively deployed in these environments.

Recent works for improving performance under power-constrained environments, compare many existing power capping mechanisms [11, 51] for a wide range of scenarios, and report that RAPL capping almost ubiquitously outperforms previous software-based capping methods. Based on this observation, several state-of-the-art works propose hybrid solutions, which combine software mechanisms with RAPL. In particular, [11] proposes PUPiL, a hybrid software-hardware framework, which uses RAPL in conjunction with a heuristic algorithm for identifying the best thread placement on cores scheme (thread-packing). Furthermore, [52] also relies on RAPL

to cap the power of the CPU and RAM. It explores how cross-component power allocation of a single node will affect the performance of multi-threaded workloads, under a constrained power budget. [53] presents a holistic methodology that utilizes the techniques used in [11, 52] synergistically to further improve the performance of parallel workloads. Moreover, [54] introduces PTRADE, a performance management framework that is general with respect to the components it manages and can be deployed to work on a new system with different components without redesign and reimplementation. Moreover, [55] integrates migration and co-scheduling policies into an operating system scheduler and into a virtualization system, allowing placement decisions to be made both within and across physical nodes, and reducing contention both for individual tasks and complete applications. [56] introduces a new metric for CPU energy performance, millions-of-instructions-per-joule (MIPJ), considers several methods for varying the clock speed dynamically under control of the operating system, and evaluates the performance of these methods using workstation traces. [57] surveys the vast field of research on energy-cognizant schedulers and discusses scheduling techniques to perform energy-efficient computation.

The previous works on power capping propose software-level optimizations, potentially in conjunction with hardware techniques such as RAPL, to improve performance in power-constrained environments. However, as shown in our investigation, CPU voltage margins reduction in the form of RVSCap results in better performance compared to the hybrid RAPL+PUPiL mechanism utilized in these works. Most importantly as shown, our approach is compatible with the above state-of-the-art solutions and could be employed in conjunction with them to provide additional performance benefits.

Chapter 4

Dynamic Reduction of Workload-Dependant CPU Voltage Margins

In this Chapter we, again, focus on the question "How to exploit CPU voltage margins reduction?". More specifically, we investigate the workload-dependability of the extent of possible voltage margins reduction and how to exploit it to further improve the power-efficiency of computing systems. However, a critical challenge is to reduce the margins as much as possible yet without compromising the reliability of the system. To this end, we have designed, developed, deployed and evaluated a run-time Extended Dynamic Voltage Scaling (xDVS) governor for off-the-self systems, which dynamically and adaptively reduces voltage margins by applying a lower, yet safe CPU supply voltage to improve power and energy consumption. The xDVS governor monitors the utilization of CPU resources and uses a prediction model to estimate and apply a new safe supply voltage to the CPU. To train the prediction model we perform an offline characterization of voltage margins on Haswell i7–4790 and Skylake Xeon E3–1220 v5 processors, by utilizing $(\mathrm{XM})^2$ (presented in Chapter 6), using a diverse set of benchmarks which stress different components of the CPU microarchitecture, such as the rate of FP computations, thermal load, and memory pressure.
The main contributions and outcomes discussed in this Chapter are the following:

1. We develop a model that takes as input selected CPU performance counters and core utilization and estimates the voltage margin of the workload on the specific CPU part for the base CPU frequency. This estimation can be exploited to safely undervolt CPUs to achieve lower power consumption and higher energy efficiency.

2. To the best of our knowledge we are the first to implement and deploy a Extended Dynamic Voltage Scaling(xDVS) governor, which, guided by such a

model, dynamically adjusts the CPU supply voltage to levels below conservative nominal values. Compared to the stock Intel (*P-state*) DVFS governor, our approach achieves energy savings up to 42% for Skylake and 34% for Haswell CPUs with negligible overhead on performance.

We do not include the X-Gene processors in this study, because of the limitations already discussed in Chapter 3. More specifically:

- The workload-dependent part of voltage margins on X-Gene processors is very narrow under higher CPU core utilization.
- The underlying hardware mechanisms introduce high latency to monitor CPU parameters and to adjust CPU configuration. As an example, for X-Gene 2 and X-Gene 3 processors, due to very long path (user-space, operating system, I^2C interface, SLIMPro controller, voltage regulator), the latency of voltage adjustment is $1.193sec$ and $0.906sec$, respectively, on average.

These limitations significantly hinder the implementation of online, workload-dependent voltage margins reduction policies on X-Gene processors.

4.1 Offline Quantification of Voltage Margins

In this Section, we quantify the minimum tolerable supply voltage V_{min} by utilizing the $(\text{XM})^2$ framework for OS-controlled execution, presented in Chapter 6. More specifically, we use both single- and multi-instance/threaded benchmarks included in the SPEC CPU2006 [58] and Parsec [16] benchmark suites, the Linpack [59] benchmark, as well as a number of stress tests (Prime95 [60], Firestarter [61], Stress-NG [62]). For Stress-NG in particular, we include all 68 sub-tests that stress the CPU. Each benchmark is executed in two modes: either occupying a single core, or all cores of the target CPU. Single-instance/threaded experiments are executed once per core, with the running thread pinned on the respective core while the rest of the cores are idle. To fully utilize the CPU, multi-instance/threaded benchmarks are executed with a degree of parallelism equal to the number of cores, whereas in the case of single-instance benchmarks we achieve full utilization by co-executing as many copies of the benchmark as the number of cores. We refer to the combination of benchmark versions and core mappings as *configurations*. We deem a level of voltage reduction as safe, for a given *configuration*, when the corresponding benchmark executes correctly for 10 consecutive times and there are no SDCs, MCE errors, and system crashes. As an additional robustness test for the value of V_{min}, we validate

each *configuration* 1000 more times, at the corresponding level of voltage reduction, and verify that there is none of the aforementioned erratic behavior.

We perform the experimental analysis on six workstations, four featuring an Intel Skylake Xeon CPU called *Skylake 1 - 4*, and two featuring an Intel Haswell i7 CPU, called *Haswell 1, Haswell 2*. All workstations run Ubuntu 16.04LTS with Linux Kernel version 4.10.0-38-generic. Table 2.1 outlines the characteristics of each workstation as well as the nominal supply voltage for both architectures under maximum utilization.

As we discussed in Chapter 2, the reduction of CPU operating voltage for Intel processors is offset-based. Figure 4.1 illustrates the experimentally identified $\text{MSR}_{\text{offset}}$, which when applied results in the corresponding V_{min}, for the four Skylake and two Haswell CPUs, running 34 benchmarks. Since the SPEC CPU2006

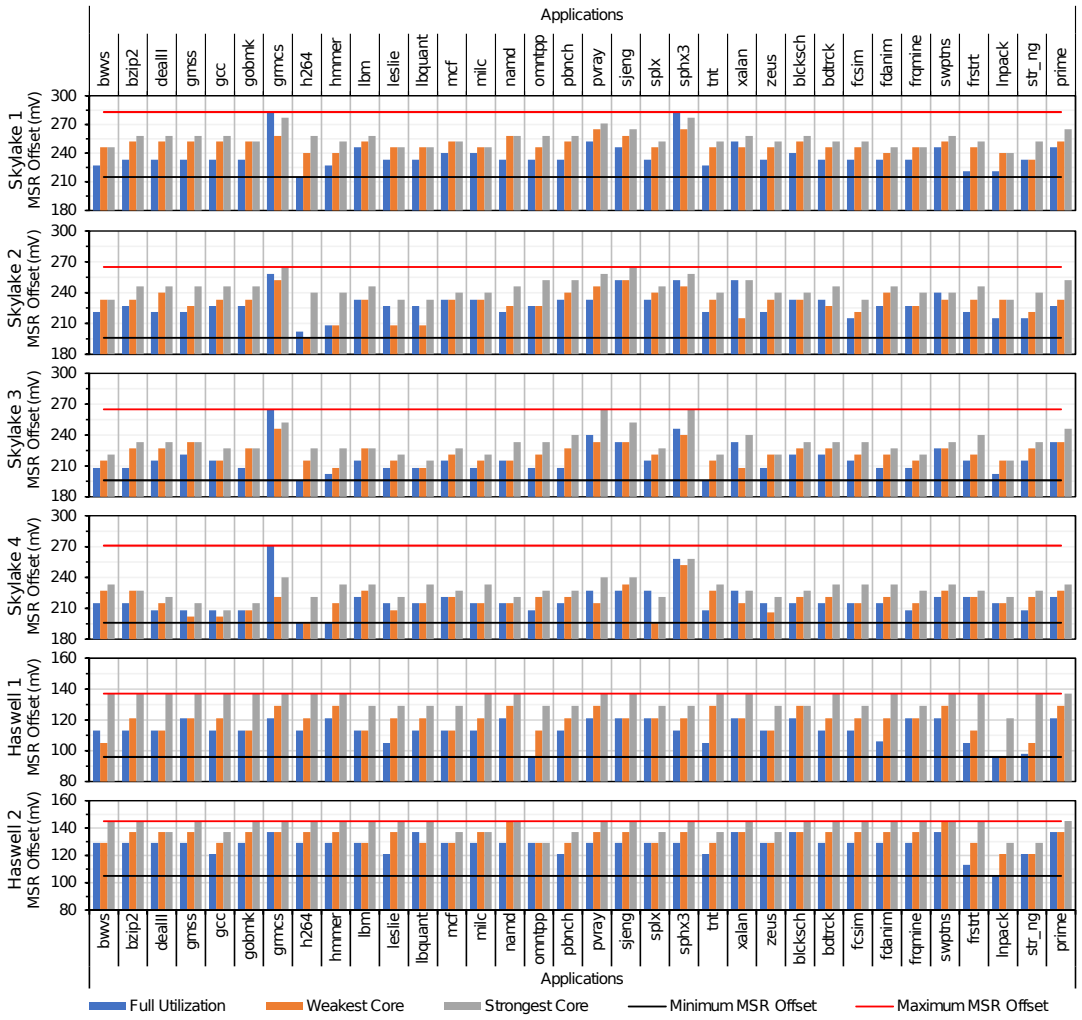


Figure 4.1: Evaluation of $\text{MSR}_{\text{offset}}$ settings for 34 benchmarks (10 runs each) in each workstation; the higher the bar, the wider the exploitable voltage margin. The horizontal dotted lines show the maximum (red) and minimum (black) values of $\text{MSR}_{\text{offset}}$.

benchmarks are single-instance, the respective margins for the fully utilized CPUs are determined by executing simultaneously four instances of the benchmark on each 4-core CPU. Voltage margins span from 17% to 24% and 9% to 13% of the nominal V_{dd} for the Skylake and Haswell microarchitectures, respectively. The difference between min and max voltage margin values (7% and 4% of nominal V_{dd} for Skylake and Haswell, respectively) is the workload-dependent margin.

Figure 4.2 shows that in the large majority of the experiments, multi-instance/threaded benchmark executions have narrower margins than when running the benchmarks in a single-threaded/instance configuration. This observation emphasizes the importance of using both single- and multi-instance/threaded programs to correctly assess V_{min}.

Moreover, unlike previous studies on ARMv8 [22] and Itanium [63] CPUs that revealed intermediate voltage regions of unsafe operation where indications of erratic behavior may be observed, for the architectures investigated in this study the transition to unreliable operation when voltage drops below V_{min} is abrupt and always leads to crashes. Even in the few cases where *SDCs* or *MCE* errors were observed, these errors were accompanied by an immediate system or application crash.

We also observe margin variations across different cores of the same part (the difference between margins of the strongest and weakest core of each CPU). The V_{min} variation when executing the same single-instance benchmarks with different cores can reach up to 45mV and 32mV for Skylake and Haswell, respectively. In Figure 4.3 we present the percentage of times each core was ranked as the weakest in terms of voltage margin when executing a single-instance benchmark. In contrast to the findings of the characterization of ARMv8 and Itanium, the CPUs in this study do not exhibit the pattern of a consistently weakest and a consistently strongest core; in our case, the strongest/weakest core varies across chips and even on the same chip

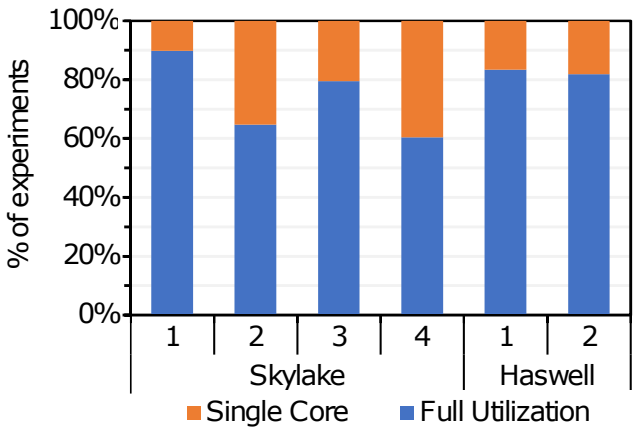


Figure 4.2: Percentage of experiments for which single core, or multi-instance/threaded workloads resulted to narrower voltage margins.

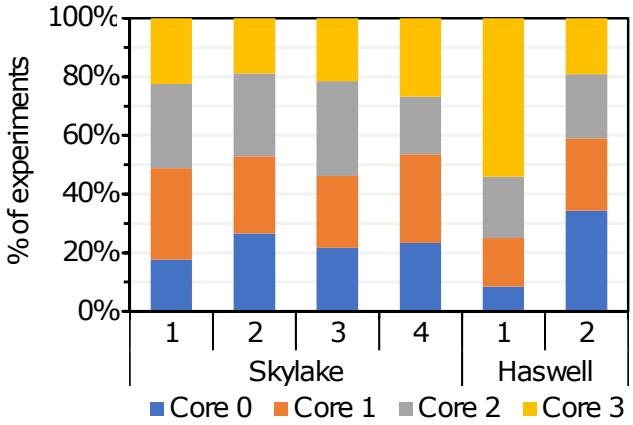


Figure 4.3: Percentage of experiments in which any given core resulted being the weakest during our characterization phase.

across benchmarks (Figure 4.3).

Figure 4.4 shows the cumulative distribution function (CDF) of the average (across all *configurations*) failure probability of each CPU, as a function of the applied $\text{MSR}_{\text{offset}}$. Note that lower slope CDF curves indicate a broader range of undervolting opportunities, depending on the characteristics of the workload and the resource pressure exercised. For example, *Skylake 4* offers an $\text{MSR}_{\text{offset}}$ range between 196mV to 271mV in which different benchmark configurations will run successfully. On the contrary, *Haswell 2* has a narrower dynamic range (105 to 145mV) exhibiting a stepwise behavior. All four parts of the Skylake family have similar margins, with *Skylake 1* being able to operate at lower V_{dd} values than the rest.

Moreover, we identified the voltage margins for a range of operating frequencies, for both CPU architectures. Figure 4.5 shows that for each frequency there is a workload-independent (green part, static) and a workload-dependent (yellow part,

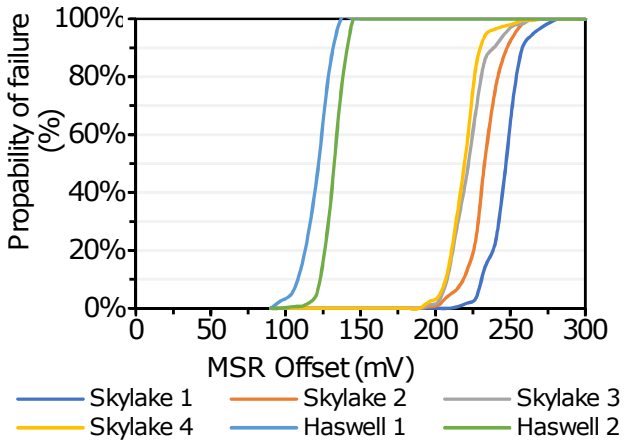


Figure 4.4: Average (across all configurations) failure probability CDF for each CPU, with respect to the applied $\text{MSR}_{\text{offset}}$.

dynamic) voltage margin which allows CPU voltage reduction without any crashes or SDCs. The static margin corresponds to a voltage range where no applications fail. The dynamic margin, in turn, corresponds to a voltage range where some – yet not all – applications fail during the characterization. The workload-dependent part of the margin increases (albeit slightly) at higher frequencies. Specifically, for Skylake 2 at 3.0GHz, the total margin is 268mV (24.5% of the nominal supply voltage), out of which 62mV is attributed to the workload-dependent margin. On the other hand, at 1.0GHz frequency, the margin is 199mV (27.5% of the nominal voltage), but the workload-dependent range is only 4mV (1.9%). Similarly, for the Haswell 2 at 3.6GHz, the total margin is 145mV (14.4% of the nominal supply voltage) and the workload-dependent margin is 52mV. At 1.0GHz frequency, voltage margin is 318mV (39.7% of the nominal voltage), but the workload-dependent range is only 10mV (3.1%).

The Skylake family exhibits wider margins compared with the Haswell family, by 103mV on average. Essentially, CPUs are black boxes and the observed nominal settings and voltage margins are a result of physical limitations (geometry), architectural limitations, designer decisions, targeted market, etc. More specifically, as shown in Figure 4.5, at 3.0GHz frequency with nominal voltage settings, Skylake operates at 1078mV and Haswell at 960mV. As a result, the reduction of voltage margins (MSR_{offset}) is lower for the Haswell CPU but in terms of the lowest subnominal voltage, the differences are not that profound.

4.2 Voltage Margins Modeling and Estimation

Microarchitectural events that disrupt the flow of pipeline execution and influence pressure on CPU resources significantly affect the minimum CPU supply voltage

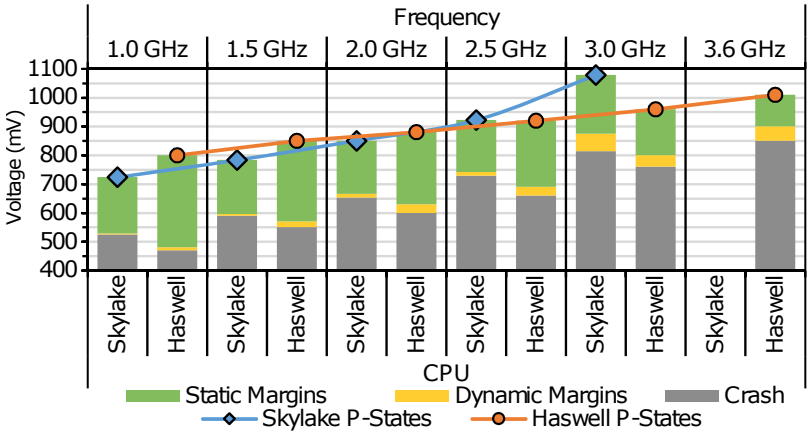


Figure 4.5: Voltage margins for a range of CPU operating frequencies for Skylake 2 and Haswell 2.

(V_{min}) required for error-free execution [64]. As software dynamically interacts with the underlying hardware, such events appear at different intensities, therefore the effective V_{min} may vary during execution (dynamic voltage margins). This presents opportunities for dynamic, workload-aware undervolting at execution time.

This Section introduces the methodology we used to build a model that predicts an operating voltage V'_{dd}, which is lower than the nominal supply voltage V_{dd} but safe, i.e., higher than V_{min} for the specific combination of workload characteristics, execution configuration, and CPU part. The four steps of our approach are the following:

1. We determine the V_{min} for different cores (and all cores together) of the target CPU part for different workloads via offline characterization, as discussed in the previous Section;

2. We profile the same workloads to quantify their interaction with the CPU using online performance counters;

3. We combine V_{min} characterization and the profiling data to fit two models (based on Random Forest Regression [65]) one for single-instance execution and one for multi-instance/threaded execution that dynamically predict a safe supply voltage V'_{dd};

4. and finally the models are used by a dynamic voltage scaling governor to dynamically adjust the supply voltage based on the models' predictions as well as the resource pressure of the CPU.

Note that this can be relatively time-consuming (in our case, the characterization took two days) and thus may not be affordable during the CPU production stage. However, in HPC and cloud environments it could be performed as part of the deployment process before a node becomes operational in the infrastructure.

4.2.1 Profiling

During profiling, we use a set of performance metrics observable through the respective PMU counters to quantify the utilization of and pressure to microarchitectural components. Intel x86-64 PMUs can only track a limited number of performance counters at the same time (in the CPU families we use in this study, up to 8 per core). We consider 85 and 80 performance metrics for Skylake and Haswell, respectively. We include all the metrics used by Intel's Top-down Microarchitecture Analysis Method (TMAM) [35], as well as additional metrics that capture operational aspects of the system, such as the CPU operating temperature, CPU power consumption, and DRAM power consumption. To collect data for all respective metrics, we

perform multiple executions for each benchmark configuration, and in each execution, we record a subset of performance counters until all metrics are covered. We sample the counters every 100ms using Linux *Perf* tools [66]. As we discuss later, we reduce the number of metrics used for model training to a maximum of 8, so we can sample all corresponding performance counters in a single execution at xDVS deployment time.

4.2.2 Model type

Our goal is to estimate a safe MSR_{offset} according to the resource utilization and pressure quantified by the performance metrics. The values obtained by offline characterization do not always exhibit a simple, monotonic behavior concerning the obtained performance metrics. Consequently, linear regression models do not adequately capture the MSR_{offset} of the applications. Instead, to predict MSR_{offset} as a combination of the aforementioned metrics, we employ a machine learning ensemble technique, called Random Forest Regression (RFR) [65]. A random forest is a collection of regression decision trees, each used to independently predict a value based on an input vector. The model predicts by averaging over the predictions of all regressions trees.

Due to the limitations of Intel PMU, at execution time we are limited to concurrently measuring up to 8 PMU events per core. For our model to be applicable in an online manner, the number of performance metrics/features needs to be reduced. This also avoids overfitting and decreases the runtime overhead of the model. To reduce the number of performance metrics, we rank their importance by estimating their mutual information (MI) for the MSR_{offset} target variable. MI is an algorithm commonly used for feature selection in machine learning [67]. It ranks different features by assigning weights so that the higher the weight the more important the feature for modeling. The algorithm assigned the highest importance to the metrics listed in Table 4.1 in decreasing order of importance. Note that the highest-ranked metrics essentially characterize the instruction mix of the workload.

During the offline profiling phase, the performance metrics are collected through multiple executions for the same configuration of each experiment. After the MI ranking step, we repeat the experiments so that the selected metrics (Table 4.1) are collected during the same execution. These data are normalized to take values between 0 and 1. As a normalizer, we use the sum of all available slots during the sampling period, which is equal to the number of pipeline slots (4 in our architecture) multiplied by the number of clock cycles (the respective counter does not fall into the limitation of 8 PMU events per core).

Skylake	**Haswell**
UOPS_DISPATCHED.PORT_0:	*IDQ.ALL_DSB_CYCLES_4_UOPS:*
Uops dispatched for execution in port 0 (Port 0 is responsible for Int., FP, vector ALU, mult, div and branch operations).	Cycles in which Decode Stream Buffer (DSB) is delivering 4 Uops.
UOPS_DISPATCHED.PORT_4	*UOPS_EXECUTED.PORT_0:*
Uops dispatched for execution in port 4 (Port 4 is responsible for Store operations)	Uops dispatched for execution in port 0 (Port 0 is responsible for Int., FP,vector ALU, mult, div and branch operations).
UOPS_DISPATCHED.PORT_1:	*UOPS_EXECUTED.PORT_1:*
Uops dispatched for execution in port 1 (Port 1 is responsible for Int., FP and vector ALU operations).	Uops dispatched for execution in port 1 (Port 1 is responsible for Int., FP and vector ALU operations).
UOPS_DISPATCHED.PORT_5	*UOPS_EXECUTED.PORT_5:*
Uops dispatched for execution in port 5 (Port 5 is responsible for Int. and vector ALU operations).	Uops dispatched for execution in port 5 (Port 5 is responsible for Int., vector ALU operations).
UOPS_DISPATCHED.PORT_2	*UOPS_EXECUTED.PORT_2:*
Uops dispatched for execution in port 2 (Port 2 is responsible for Load operations).	Uops dispatched for execution in port 2 (Port 2 is responsible for Load operations).
EXE_ACTIVITY.PORTS	*UOPS_EXECUTED.PORT_6*
Cycles for which one uop began execution on any port, and the Reservation Station was not empty.	Uops dispatched for execution in port 6 (Port 6 is responsible for Int., and branch operations).
UOPS_EXECUTED.THREAD	*UOPS_EXECUTED.PORT_3*
Number of Uops executed by this hardware thread.	Uops dispatched for execution in port 3 (Port 3 is responsible for Load operations.)
MEM_UOPS.ALL_STORES:	*MEM_UOPS.ALL_STORES:*
Number of store operations retired.	Number of store operations retired.

Table 4.1: Most influential performance metrics for V_{min}, as ranked by the MI algorithm.

Finally, recall that $\text{MSR}_{\text{offset}}$, besides being dependent on the workload, also depends on the resilience of the specific cores (inter-core variation) and degree of CPU utilization. Moreover, in a realistic scenario, the operating system may, independently of our mechanisms, modify the topology of active cores via thread migration. Our model should provide a safe setting irrespective of workload mapping to cores. To capture these variations, we construct two models for each CPU part. One model covers the case of single-core execution (single-core model) and is trained using the $\text{MSR}_{\text{offset}}$ of the weakest core for each benchmark configuration. The other model covers the case where multiple cores are occupied (multi-core model) and is trained using the data of the benchmark configurations that use all cores of the CPU. So, in total, we build 12 different models (6 CPU parts, each having 2 models for single- and multi-core CPU utilization).

4.2.3 Model training

Most applications exhibit different execution phases in terms of CPU resource pressure and performance characteristics. For example, Figure 4.6 shows the number of uops dispatched for execution in port 1 for the first 30 secs of execution for two applications with a single-phase (*swaptions, hmmer*) and one application with multiple

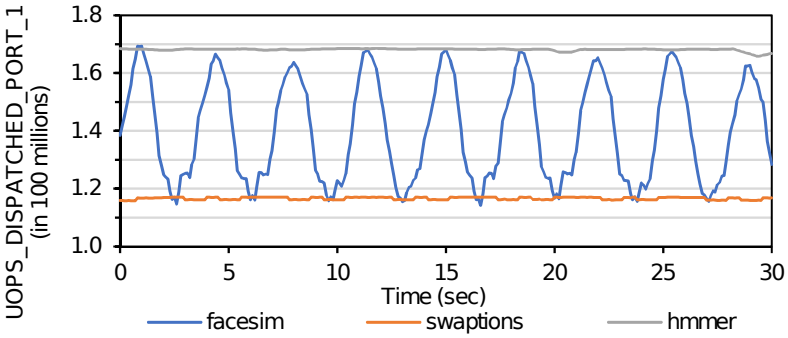


Figure 4.6: The number of dispatched uops in port 1 during the execution time of an application.

phases (*facesim*). However, our offline characterization determines the single worst-case V_{min} across all execution phases of an application. This can negatively affect the effectiveness of training our model as it will overgeneralize, trying to correlate wildly varying performance counter patterns with the same V_{min}.

To provision for such cases, we bias the training input set to include mainly applications with few execution phases such as *hmmer* and *swaptions*. We first rank the 34 benchmarks according to the number of phases they exhibit, normalized to their execution time. Phase change detection is performed by monitoring large changes (more than 10%) on any of the Level-1 performance metrics of TMAM [35]. The smaller the number of phase changes, the higher the ranking of the application. Then, we select the top 90% (most stable) applications for training and validation. 90% of the selected applications are used for training and 10% for validation. The remaining 10% of applications (the ones with the largest number of phase changes) serve as the testing (evaluation) set. The validation set includes *bodytrack*, *freqmine*, *gcc*, and the testing set includes *facesim*, *zeusmp*, *fluidanimate*, *stress_ng*. Figure 4.11 validates our training decision, as the model can correctly generalize and predict different phases of an application even though these applications are not included in the training procedure. Note that the $\text{MSR}_{\text{offset}}$ changes of *facesim* in Figure 4.11 are similar to the phase changes of Figure 4.6.

The training input set, which consists of samples (in 100ms sampling intervals) of the eight performance counters (Table 4.1) of the applications selected for training, is used to train the Random Regression Forest to minimize the Root Mean Square Error (RMSE) between the predicted V'_{dd} and the V_{min} determined via the offline characterization. Typically, RFR is defined by a predefined number of different simple estimators (the decision trees) and by the maximum depth of each decision tree. Using a large number of estimators and/or using deep trees for the prediction can both incur high-performance penalties, and result in overfitting. In the end, the models that minimized the RMSE for both the training and validation set consist of only

three estimators, with the maximum depth of each estimator being equal to three. The average RMSE is 7.01mV and 5.45mV for the Skylake and Haswell families, respectively.

Over- or under-prediction is a common side-effect of many modeling approaches. In our case, over-predicting the $\text{MSR}_{\text{offset}}$ would result in reducing the supply voltage below V_{min} leading to unreliable operation. Therefore, as a last step, we introduce a small *safety margin* to the estimated $\text{MSR}_{\text{offset}}$ value. For each model, the safety margin is set equal to the RMSE between the value that is predicted for the validation data, and the $\text{MSR}_{\text{offset}}$ value that was observed during the offline characterization for the respective applications in the validation set. Equation 4.1 provides the final offset that is applied on the MSR registers of the CPU (where $\vec{X}$ denotes the input vector to the model).

$$MSR'_{offset}(\vec{X}) = MSR_{offset}(\vec{X}) - safetyMargin \tag{4.1}$$

The safety margin controls the aggressiveness of our methodology. Using very small values would result in aggressive undervolting at the risk of reduced reliability, whereas too large safety margins would merely decrease the energy gains. A conservative, yet pessimistic, safety margin is the maximum error between the predicted values and the validation data. Instead, we use RMSE as safety margin (7.01 and 5.45 mV for the Skylake and Haswell families, respectively) and trust the modeling procedure to correctly handle the outliers. Our approach is validated in the evaluation; no failures have been observed during application execution.

Figure 4.7 shows the predictions of our model for benchmarks in the validation

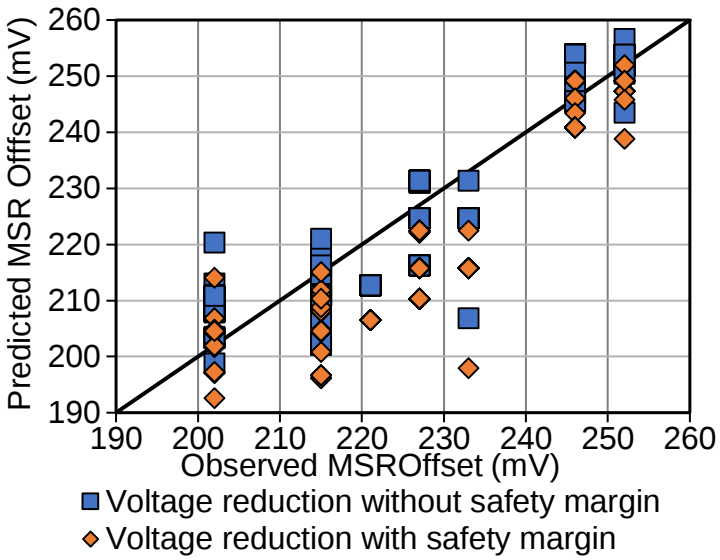


Figure 4.7: Prediction of our model with and without the safety margin, for samples in the validation data set.

set, running on the two Skylake processors, with and without the safety margin. The black line represents the $\text{MSR}_{\text{offset}}$ values that would be predicted by a perfect model (corresponding to the observed V_{min}). Predictions above the line correspond to application phases that can be executed in a V_{dd} lower than the conservative, application-wide V_{min} which was obtained in the offline characterization. Such cases are discussed in Section 4.4. Including the safety margin reduces power efficiency but enables safe operation.

4.3 Extended Dynamic Voltage Scaling

The model introduced in the previous Section can be used online, to enable fine-grained undervolting at runtime, according to the resource pressure quantified by performance counters samples and the number of cores utilized by the workload. To this end, we have implemented an extended dynamic voltage scaling governor (xDVS), which runs periodically. Upon invocation, it feeds the model with the performance counter measurements collected during the previous interval and uses the $\text{MSR}_{\text{offset}}$ suggested by the model to derive a less-than-nominal but still safe supply voltage V'_{dd}.

The xDVS governor is implemented as a Finite State Machine (FSM), depicted in Figure 4.8. The V'_{dd} selected for the next interval depends on the prediction of the model, the number of active cores and the current state of the governor. Below we describe the states and the logic of xDVS:

Back-Off: In this state, the measurements collected during the previous interval are not considered as representative for the workload during the next interval and, therefore, should not used as input for the model. This is the case, for example, when

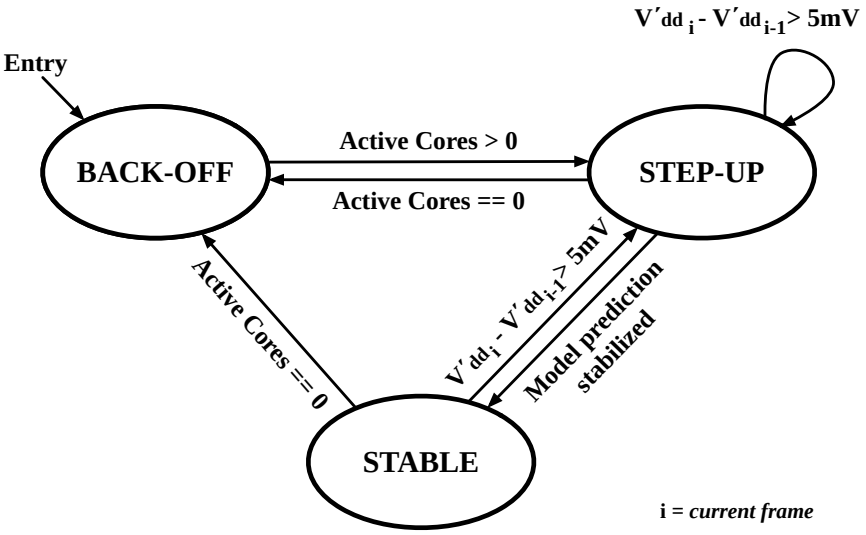


Figure 4.8: FSM diagram of the xDVS governor.

a core starts executing a new process/thread, or after a long idle period that allowed the operating system to place that core in a higher C-state (lower power mode). In this state, xDVS does not invoke the model and applies the nominal V_{dd}.

Step-Up: This state provides a smooth transition between the *Back-Off* and the *Stable* states. When in this state, the governor invokes the model to perform predictions in a way similar to the *Stable* state, but if the model suggests a large reduction to the supply voltage, this is applied gradually, in smaller steps of 5 mV. This filters abrupt and potentially risky – in terms of reliability, should the behavior of the workload change again – voltage reductions.

Stable: The governor collects performance counter values for each core and invokes the single-core or multi-core model depending on whether only one or more cores are currently active. Then, based on the suggested $\text{MSR}_{\text{offset}}$, the governor applies the new V'_{dd}.

The xDVS governor monitors which cores are active, by observing the percentage of time the system was in the $C0$ state ($C0$ residency) of each core. We consider a core as active when $C0$ residency goes above 70%, and a core is classified as inactive when the corresponding $C0$ residency drops below 50%. We avoid thresholds close to 100% or 0% as this would result in unnecessarily high transition sensitivity (e.g., a core would be considered as active when merely moving the mouse of a desktop).

When all cores are considered as inactive, the xDVS governor enters/stays in the *Back-Off* state to provide a setup time to new workloads. When there is at least one active core during two consecutive sampling intervals, xDVS transitions to the *Step-Up* state. It remains in this state until the supply voltage is gradually reduced to the level suggested by the model, and once this level is reached the governor enters the *Stable* state.

Whenever the model suggests a large increase to the supply voltage, the increase is implemented immediately (without any stepping), irrespective of the state of xDVS as this is considered an emergency and any delay could compromise the reliability of the system. Also, increasing the supply voltage does not introduce any risk if the workload suddenly changes; in the worst case, an opportunity for energy savings will be missed.

Figure 4.9 illustrates the level of undervolting applied by xDVS on Skylake 2 for an indicative scenario with two abruptly alternating workloads with significantly different margins, namely gromacs, and h264ref. We execute four instances of each of the applications to fully utilize the CPU. As discussed in Section 4.1, the CPU supply voltage can be reduced by 257mV and 198mV for gromacs and h264ref respectively. In every transition from the workload with high tolerance (gromacs) to the workload with a lower tolerance (h264ref), the level of undervolting drops immediately

to increase the CPU supply voltage. In the reverse transitions, the undervolting level gradually increases (CPU voltage decreases) until it reaches the level that is suggested by the model.

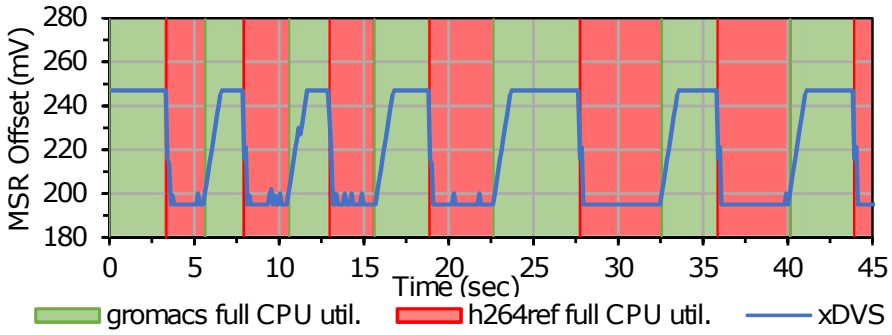


Figure 4.9: MSR_{offset} applied by xDVS on Skylake 2, for sudden transitions between workloads with different margins.

4.4 Experimental Evaluation

In this Section, we evaluate the ability of the xDVS governor to dynamically identify voltage margins based on our prediction model and drive the CPU to a more energy-efficient state. We quantify the resulting energy gains using the benchmarks in the test set (*stress_ng, zeusmp, fluidanimate,facesim*), which have not been used during model training and validation and we compare the energy gains of xDVS against the Intel *P-state* DVFS governor. We used Linux *Perf* [66] tool to monitor energy consumption.

In addition, we evaluate xDVS with 4 larger-scale applications, namely Gem5 [68], a CPU miner [69], the compilation of the Linux Kernel [70] and Polybench [71]. More specifically, Gem5 simulates an ARM processor using the system emulation mode. During the simulation, we execute a variety of simple micro-kernels such as Integer and Floating-Point Matrix Multiplication, Sorting algorithms, and Combinatoric problem-solving kernels. The CPU miner employs five different hashing algorithms (Bitcore, Sha256d, Xevan, Timetravel, and Cryptonight [72]), all used to perform mining for different cryptocurrencies such as Litecoin and Bitcoin. Finally, we use multiple solvers and stencils included in the Polybench suite such as Alternating Direction Implicit (ADI), Jacobi, LU factorization, Gram–Schmidt process, Gauss–Seidel. Each one of these 4 larger-scale applications is executed for approximately 1 hour. Figure 4.10 shows the average MSR_{offset} applied by xDVS when applications are executed on all cores, or on the weakest core. For the large-scale applications, in which there is no offline characterization, we present the average dynamic MSR_{offset} across all single-core executions.

The MSR_{offset} applied by xDVS includes the extra safety margin, thus we expect it to be, on average, more pessimistic than the MSR_{offset} identified by offline characterization (as shown in Figure 4.10). The voltage applied by xDVS (including the safety margin) is on average 9.3mV and 8.2mV higher than the one identified by offline characterization for the Skylake and Haswell microarchitectures respectively. Despite the safety margin, there are cases in which xDVS successfully identifies application phases and adjusts the supply voltage to lower values than those identified in the offline characterization, without compromising system reliability. Note in Figure 4.10 that xDVS identifies wider static and dynamic (phase-dependent) voltage margins for the Skylake family, which is compatible with the findings of the offline characterization presented in Figure 4.4.

Figure 4.11 shows the dynamically applied MSR_{offset} when four benchmarks are scheduled for consecutive execution on the same core for all Skylake and Haswell processors. xDVS can capture the dynamic nature of the applications. As an example, in *facesim* (an iterative application that executes 3 separate kernels per iteration) the MSR_{offset} varies between 201mV and 233mV, for Skylake 2, as the model captures the phases of the application. Similar behavior is observed for all Skylake processors for *facesim* application where xDVS captures the periodic phase changes and drives undervolting even more aggressively than what static MSR_{offset} dictates (gray line).

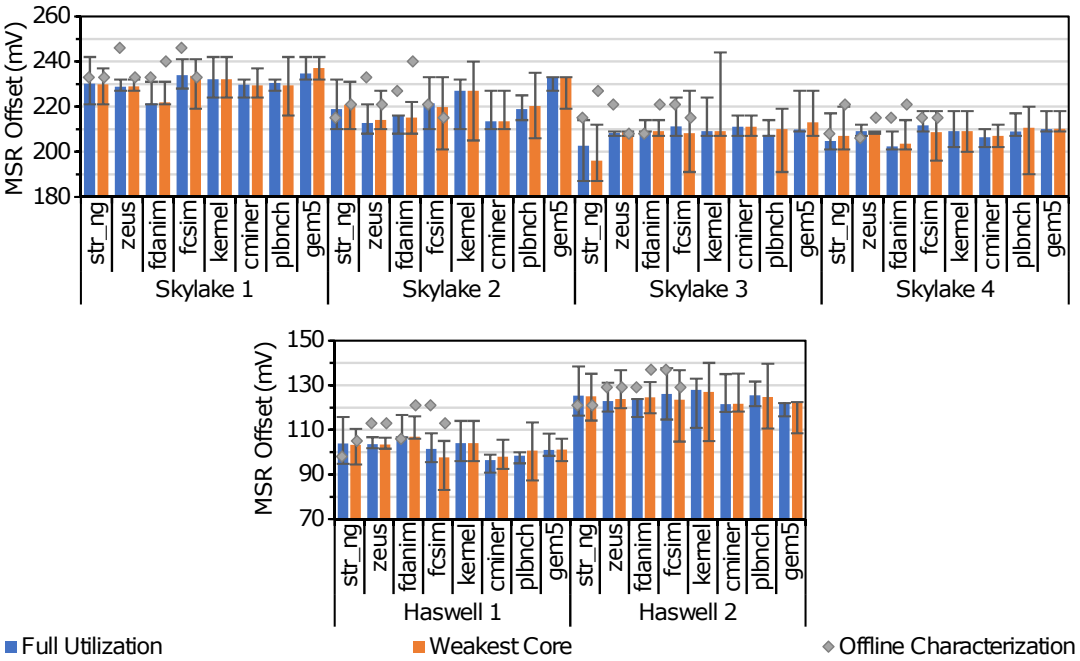


Figure 4.10: The bars show the average dynamic MSR_{offset} applied by xDVS, for Skylake (up) and Haswell (down) workstations. The min-max bars represent the minimum and the maximum MSR_{offset} applied by xDVS. The gray diamond represents the MSR_{offset} as identified by offline characterization at the granularity of the whole application.

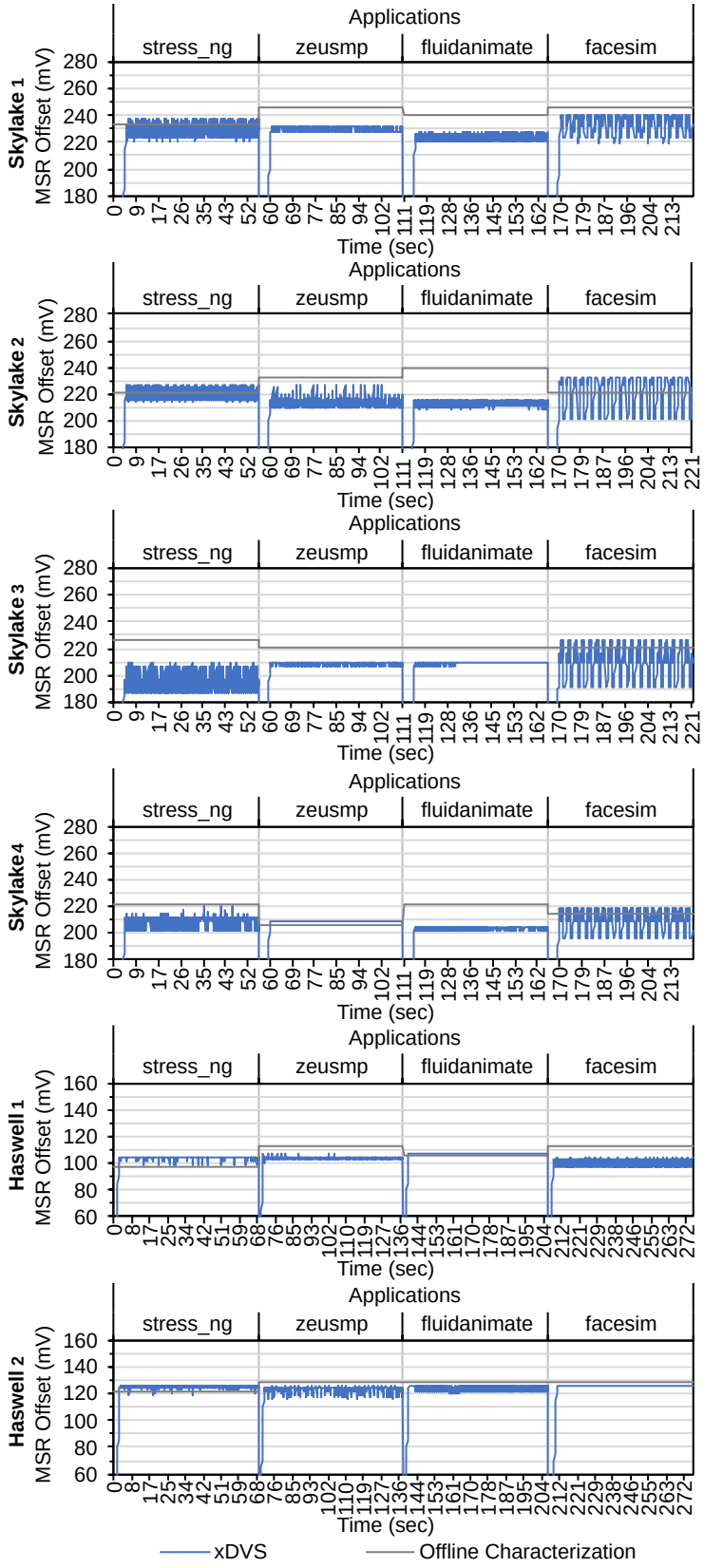


Figure 4.11: Timeline showing the MSR_{offset} for consecutive single core executions of four applications on all target parts (CPU chips).

On Haswell processors, on the other hand, the prediction model employed by xDVS does not identify such opportunities for *facesim*.

Figure 4.12 shows the MSR_{offset} timeline when the four larger-scale applications are executed on each workstation. The graphs reveal relatively large intra-family margin variations, as well as variations due to different workload characteristics. Note that the xDVS governor can capture the different algorithms consecutively used by the CPU miner application. Note also the steep MSR_{offset} drops when the OS schedules a new application, due to our xDVS governor transitioning to the *Back-Off* state. Interestingly, when running the Linux kernel compilation on the Haswell processors, the governor transitions frequently to the *Back-Off* state. The Haswell PCs are equipped with a slow Hard Drive Disk (HDD). When compiling, the large source files of the kernel and object files are read from and written to the HDD, resulting in the $C0$ residency of each core to drop below 50% (since most of the time cores wait for I/O operations). This makes xDVS to transition to the *Back-Off* state. This effect is not observed on the Skylake workstations, as they are equipped with fast Solid State Drive (SSD). Figure 4.13 shows that the xDVS governor achieves 29.59% and 21.93% average CPU energy gains and can reach up to 42.68% and 34.37% for the

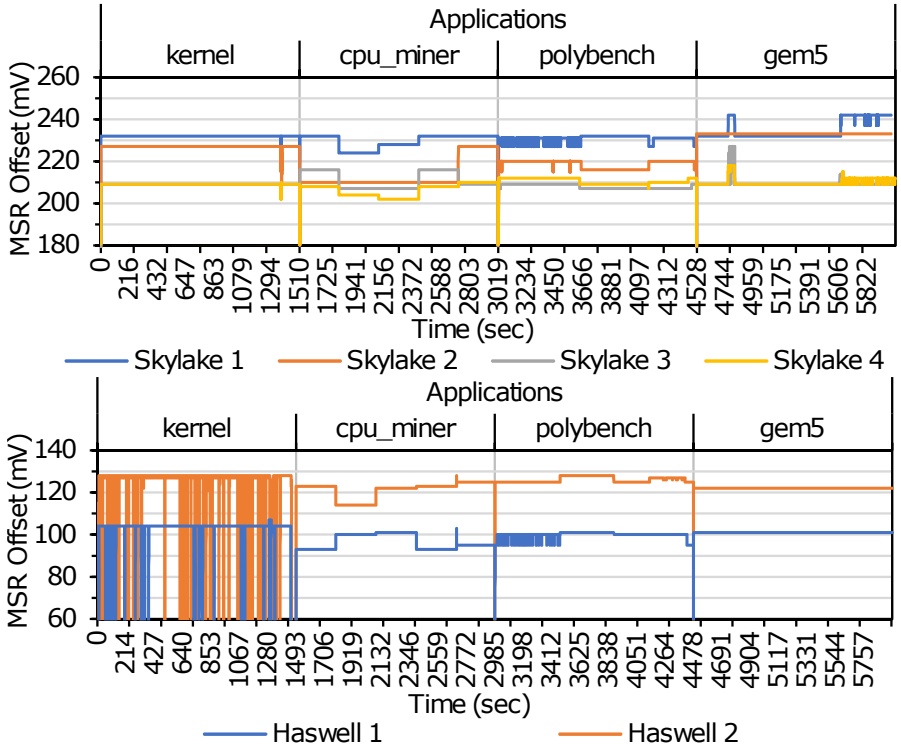


Figure 4.12: The timeline showing the MSR_{offset} applied by xDVS, while executing the large applications in full system utilization for Skylake (up) and Haswell (down) workstations.

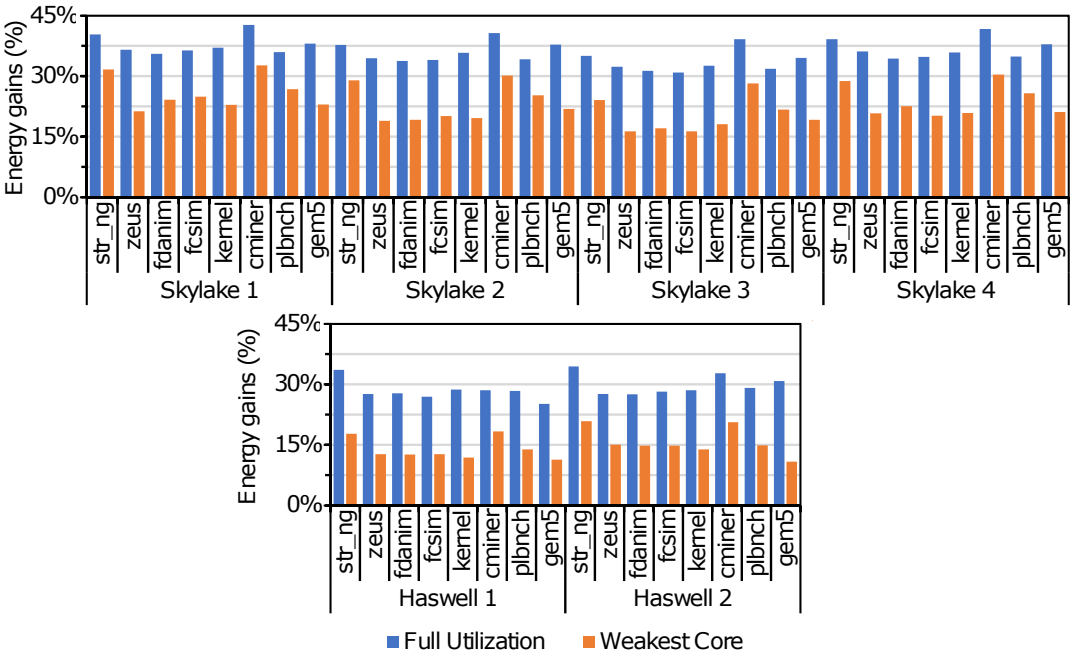


Figure 4.13: Energy gains of xDVS when compared with Intel P-state governor for Skylake (up) and Haswell (down) CPUs. The grey horizontal lines represent the $\text{MSR}_{\text{offset}}$ obtained by the offline characterization.

Skylake and Haswell processors respectively, compared with the Intel *P-state* governor. Higher gains are – as expected – obtained when cores of the CPU are highly utilized.

Note that when xDVS is enabled, Intel's Turbo Boost technology is disabled and the operating frequency is pinned to the CPU maximum nominal frequency. This incompatibility between xDVS and Turbo Boost technology is because of the way the Turbo Boost mechanism operates. More specifically, there are different Turbo Ratio limits, which translate to various CPU operating frequencies, for different CPU utilization intensities – in terms of utilized CPU cores – by the executing workload. Consequently, when Turbo Boost is enabled, the CPU operating frequency is controlled by the hardware and changes without any notification to the system software stack. The performance penalty due to disabling Turbo Boost translates to an average execution slowdown of 8.73% and 5.59% for the Skylake and Haswell processors, respectively. However, the performance overhead of the xDVS governor itself is minimal. When the governor is active and the prediction requires only $160ns$, while changing the new $\text{MSR}_{\text{offset}}$ requires 155us. On average, the performance overhead is equal to merely 0.04% of execution time when xDVS operates at a 100ms interval. If the interval is set to 10ms (lowest supported sampling interval of performance counters through *Perf* API), the overhead remains at 0.04% for single-core utilization but increases up to 3.5% on average at full CPU utilization. Even in this case, xDVS still

provides significant energy gains compared to the Intel P-State CPU governor.

4.5 Related work

In contrast to our work, all previous efforts in predicting voltage margins using performance counters only report theoretical energy gains, without deploying their model. They rely only on theoretical results which, if not validated in real hardware, could result in unreliable operation and even system crashes. Moreover, they do not project the potential energy gains, as this work does, in large scale deployments, for which a fault tolerance mechanism – such as checkpointing – is necessary. Our FSM based governor uses performance counter values to reduce the voltage margin of the system at run time on real hardware for unseen workloads which consist of several different applications executing concurrently.

In particular in [22], due to the manifestation of SDCs before system crashes, the authors propose a severity function that can predict safe, SDC-free undervolt levels for each core of the processor. Based on this function and the corresponding core V_{min} resulted from the offline characterization, they produce a linear regression model that predicts the V_{min} of a core for a single-instance workload. Their model is trained and evaluated using only single-core executions. In contrast to their work, we use multi-instance/threaded workloads and demonstrate their importance as multi-instance/threaded workloads usually result in more conservative V_{min}. Moreover, [22] states that V_{min} prediction is uncorrelated to performance counters and in combination with limited dynamic variation of V_{min} in ARMv8 a naive prediction of using average values is sufficient. In a more recent paper [73], the same authors present a comprehensive statistical analysis for the same platform that improves prediction on dynamic variations. Note also that we identify margin deviations between off-the-shelf, commercially available parts, and not parts from extreme corner nodes (TTT, TFF, and TSS in [22]).

In [74] the authors characterize the voltage margins of two x86-64 microprocessors (Sandy-Bridge-E and Haswell) for a subset of the SPEC CPU2006 benchmark suite. Similar to [22], they do not consider the implications induced by multi-instance/threaded executions. We show that multi-instance/threaded executions significantly limit the depiction of the whole voltage margins spectrum. Also, in our work, we quantify the voltage margin deviations between off-the-shelf parts that belong to the same CPU family and eventually deploy a governor that exploits effectively the voltage margins.

The heuristics presented in [63] and [75], that dynamically reduce voltage margins while always preserving safe operation, are based on the error correction ECC

hardware built on modern processors such as the server-class Intel Itanium 9560. A key observation here is that as V_{dd} is lowered, ECC correctable errors appear before uncorrectable errors (SDCs and CPU crashes). The rate of ECC correctable errors is used as an indicator of how to adjust the V_{dd} voltage. In our work, errors reported by the ECC mechanism appear very rarely, and they are always accompanied by immediate CPU crashes. We do not rely on ECC, but rather predict a safe supply voltage using a selected set of performance counters as estimators. Eventually, the methodology we propose is generic enough that can be applied to any processor, that provides the ability to manipulate the supply voltage, by measuring performance counters.

Authors in [76] exploit the voltage margins of a server-class 8-core Power7+ processor. In contrast to our work, this approach utilizes specialized hardware mechanisms such as critical path monitor sensors and digital phase-locked loops, to detect and exploit the margins. As the number of utilized cores increases by the executing workload, the larger the IR drops are across the chip. The power gains obtained by their adaptive margin scheme begin to diminish as the executing workload utilizes more CPU cores. On the contrary, the CPU families under investigation in our work, exhibit the exact opposite trend, where greater power improvements are observed on multi-threaded/instance workloads. More importantly, our xDVS governor can exploit voltage margins regardless of the CPU core utilization scheme.

A study of the voltage margins on several Kepler and Fermi GPUs is presented in [77]. They first characterize the impact of the process, temperature and voltage variation on V_{min}, and then predict safe values of V_{min} by deploying a linear regression and a neural network model. They show that high energy margins can be achieved by shaving conservative guardbands in modern GPUs. Our work targets CPU architectures which are significantly more complex than GPUs. Moreover, typically CPUs - contrary to GPUs - serve volatile workloads, with diverse characteristics, consisting of mixed user and OS jobs. Our model can provide accurate voltage margins predictions for workloads which consist of a mixture of different benchmarks.

Based on microarchitectural events (such as branch mispredictions and cache misses) that flush and stall the CPU pipeline and cause large voltage droops, in [64] they propose a voltage emergency predictor that learns the signatures of such voltage emergencies and triggers a CPU throttling mechanism (e.g. increase voltage or decrease frequency) to prevent their recurrence. This work is based on CPU modeling on the Simplescalar simulator. In a subsequent paper, the authors measure run time voltage emergencies on a Core 2 Duo processor and attribute them to pipeline stalls and flush [78]. Based on these experimental observations, they propose that a mechanism that uses more aggressive margins and a recovery (check-point based)

scheme may be better than a fail-safe static margin. Moreover, they propose a program scheduling mechanism so that the combined voltage droop is canceled out as much as possible. Unlike our work, the previous papers do not resort to undervolting to reduce voltage margins but instead, they describe techniques and provide solid design guidelines that could be exploited by the supplementary mechanism we describe in Chapter 5.

Several proposed techniques present design approaches at the circuit or microarchitectural level that trade reliability for lower voltage, by attempting to reduce the voltage down to the point that produces maximum allowable errors without causing catastrophic failures [79]. Several approaches propose methods that ensure the correct operation of caches under undervolted conditions at the microarchitectural level [80, 81, 82]. Architectural techniques are presented to eliminate data corruption, and by extension enable cache operation at scaled voltage settings. In our work, we are only interested in the behavior of the whole CPU, and not of any specific component. The Razor processor is designed with builtin support for dynamic detection and correction of timing failures of the critical paths [83]. EVAL is a framework for dynamic adaptation of supply voltage, processor frequency and body bias using a machine learning algorithm [84]. Similar ideas include dynamic pipeline adaptation transferring the time slack of faster pipeline stages to the slower ones (ReCycle)[85], and using variable latency techniques to mitigate the impact of variations on the register file and execution units in a microprocessor [86].

Furthermore, many approaches employ simulations and modeling techniques to provide design guidelines for future hardware. Authors in [87] propose a multi-core processor that can scale its resources and the number of operating cores to lower than that of integrated to meet certain power constrains. Several approaches [88, 89, 90] employ regression analysis to map certain performance counters to microarchitectural events and power-delivery estimation. [91] explores ECC protection mechanisms to enable low-power caches through a detailed SRAM failure modeling. [92] introduces a workload-dependent technique to identify the paths excited by individual applications on ultra-low-power microprocessors and reduce voltage to a level that meets the timing of those paths (instead of all paths).

Chapter 5

System Reliability when Operating at Reduced Voltage Margins

In this Chapter, we focus on the question “Is the system reliability affected?” by the operation of the CPU at reduced voltage margins. The reduction of voltage margins, even when performed in an educated and careful manner, may affect the resilience of the CPU to errors. This would, in turn, make the system more prone to failures, reducing the Mean Time Between Failures (MTBF). Both methodologies presented in Chapter 3 and in Chapter 4 operate the CPU at hand at subnominal voltages. Although we have not observed any crashes and SDCs in our experiments, voltage underscaling inevitably comes at the cost of increased probability of system failure compared to nominal CPU operation (where malfunctions occur too, yet with lower probability).

We discuss the organization and results of a long-running campaign that was conducted to validate the robustness of the mechanisms, presented in Chapter 3 and Chapter 4. Based on the results of the validation, we statistically estimate the effect of operation at reduced voltage margins on the MTBF at different confidence levels and system scales. Finally, we investigate whether the energy gains due to operation at reduced voltage margins outweigh the extra cost – due to the potentially increased MTBF – of fault tolerance mechanisms such as checkpointing in large-scale deployment scenarios.

The main contributions and outcomes of our work are the following:

1. To the best of our knowledge, this is the first work that experimentally evaluates the effects of CPU operation at reduced voltage margins with long experiments and statistically estimates the respective effects on large-scale deployments.

2. We estimate the expected energy gains of CPU operation at reduced voltage margins when taking into account the extra cost of a fault-tolerant mechanism, such as checkpointing and restore.

5.1 The Tradeoff of Operating CPUs at Reduced Voltage Margins

5.1.1 Validation of Reduced Voltage Scaling power Capping (RVSCap) and Extended Dynamic Voltage Scaling (xDVS)

To assess the risk of CPU operation at reduced voltage margins, we performed a long experimental evaluation on 16 identical Xeon E3 systems. During this campaign, RVSCap was constantly enabled on all systems, and random power caps in the range $[15W, 75W]$ (80W is the TDP of this CPU) were enforced on randomly selected combinations of applications from our benchmark set (Table 3.1). Also, we repeated the same experiment for xDVS governor. More specifically, xDVS was enabled on all nodes while executing randomly selected combinations of applications from our full set of benchmarks (presented in Section 3.3).

Both experiments were concluded, independently, after a total of 8832 device-hours, 552 hours (23 days) per system, without any observed failure such as crash, silent data corruption (SDC), or any detected error. We opted to end the experiments at this particular time point and provide a pessimistic lower-bound of the systems MTBF. However, these experimental results provide a solid indication that our mechanisms can exploit the CPU voltage margins reduction without limiting the reliability of operation of the systems.

5.1.2 Effect of Operation at Reduced Voltage Margins on MTBF

Based on the results discussed in section 5.1.1 and in order to provide a pessimistic estimation on the MTBF of systems that operate with RVSCap and xDVS, we assume that 1 system out of the 16 experiences a transient error at 552 hours.

Prior studies show that the MTBF of individual nodes in large-scale facilities can be captured by a Weibull distribution with decreasing hazard rate (DHR, shape parameter $\kappa = 0.7$) [93, 94, 95, 96]. In particular, we use the following equation

$$\text{MTBF} = \frac{1}{\lambda}\Gamma(1 + \frac{1}{\kappa}) \tag{5.1}$$

where Γ refers to the gamma distribution, λ and κ are the scale parameter and the shape parameter of the Weibull distribution law, respectively. When $\kappa = 1$ the distribution is exponential with a constant failure rate and when $\kappa = 0.7$ the distribution transforms to DHR. We note that Weibull with DHR takes into account infant mortality phenomena, which is consistent with our observations during the characterization

process of the Xeon E3 CPUs where the selection of CPU voltages below V_{min} led to failures almost instantly.

In order to calculate the scale parameter (λ), we solve the following Weibull Cumulative Distribution Function (CDF) equation for λ

$$\text{F(t)} = 1 - e^{-(\frac{t}{\lambda})^{\kappa}} \tag{5.2}$$

where, based on the results of our experimental analysis, t equals to 552 hours, κ (shape parameter) equals to 0.7 and $F(t = 552 \text{ hours})$ equals to 6.25% because we assumed 1 system failing out of the total of 16 systems.

Furthermore, given the limited number of systems tested, we apply the chi-squared distribution to extrapolate and calculate the most pessimistic (minimum) MTBF for different confidence levels (CLs) [97, 98] based on the following equation

$$\text{MTBF}_{\text{CL}} = \frac{2 \text{ MTBF}}{X^2_{1-CL}(2n+2)} \tag{5.3}$$

where X^2 refers to the chi-square distribution, MTBF is the experimentally observed MTBF and n the number of failures. To this end, we assume a distribution with 2 degrees of freedom ($n = 0$) [97, 98], since we did not observe any errors during the testing. Eventually, based on the results of our risk assessment analysis, for a confidence level of 90% $MTBF_{0.9} \geq 634.79$ days. Note that it is common for industry assessments to provide MTBF at a 60% CL [99] which in our case is $MTBF_{0.6} \geq 1595.65$ days.

5.1.3 Effects of Operation at Reduced Voltage Margins in Large, Scale-out Deployments

Assuming a platform that employs N nodes to execute a workload, the respective $MTBF_{plat} = MTBF_{node}/N$, where $MTBF_{plat}$ and $MTBF_{node}$ are the MTBF values for the whole platform and the node respectively, for any continuous failure distribution [93], as is the case with DHR Weibull.

To deal with the increased probability of failure, large-scale systems typically perform checkpointing. Given that such mechanisms introduce overhead, we evaluate whether energy savings achieved thanks to operation at reduced voltage margins outweigh this overhead. Although there are more sophisticated checkpointing mechanisms [93], we pessimistically assume a blocking, coordinated checkpointing scheme that is performed at the optimal period to capture the impact of voltage margins reduction even in such inefficient implementations. More specifically, we use

the following equations of the analysis in [93] which estimate the overhead (WASTE) and the optimal checkpointing period (T_{opt})

$$\text{WASTE} = 1 - \left(1 - \frac{C}{T}\right)\left(1 - \frac{1}{\text{MTBF}_{plat}}\left(D + R + \frac{T}{2}\right)\right) \tag{5.4}$$

$$T_{opt} = \sqrt{2(\text{MTBF}_{plat} - D - R)C} \tag{5.5}$$

where C is the checkpointing cost, T is the period of checkpointing, R is the cost of restoring the checkpoint, D is the downtime.

Figure 5.1 illustrates the energy benefits to be expected for a power capped execution at reduced margins with checkpointing vs. an execution at nominal CPU settings and no checkpointing. We should note here that, when executing at large-scale, nodes do fail and checkpointing/restart provision is necessary even when the CPU operates at nominal settings. By comparing with nominal executions without checkpointing we overestimate the protection overhead due to CPU operation at reduced voltage margins. Therefore, the overall energy gains reported in the following paragraphs are conservative.

Without loss of generality, we choose a constant $D = 2$ minutes and we let $C = R$ vary from 5 up to 60 minutes. The dashed black line represents the energy gains when operating at reduced margins with the most relaxed power cap but without any checkpointing and assuming no failures (infinite MTBF). We do not show a comparison for more restrictive CPU power caps, as these anyway result in higher energy gains (see Section 3.3).

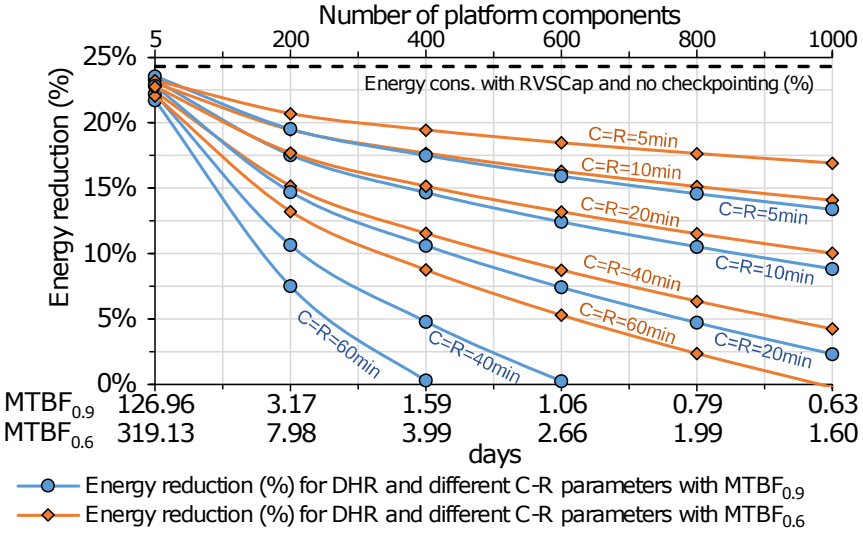


Figure 5.1: Energy reduction of RVSCap with checkpointing vs. execution at nominal settings without checkpointing, for $MTBF_{0.9}$, $MTBF_{0.6}$ and different C-R parameters.

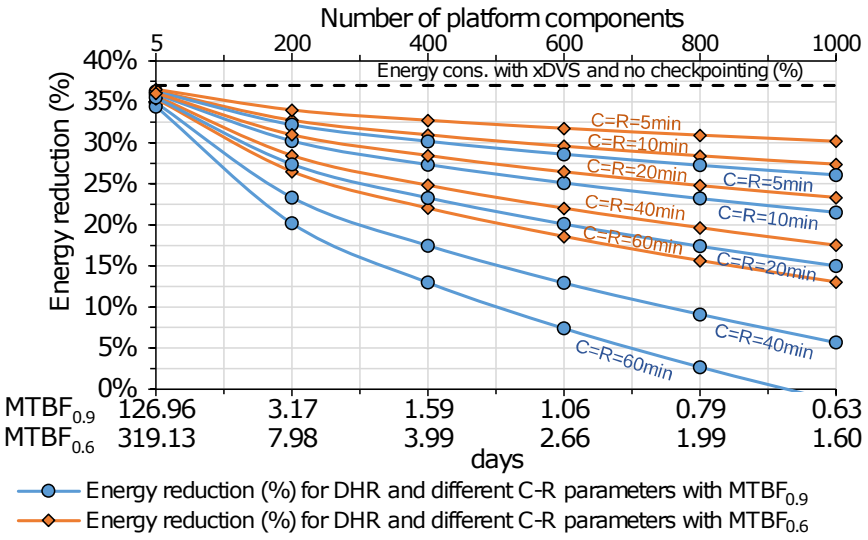


Figure 5.2: Energy reduction of xDVS with checkpointing vs. execution at nominal settings without checkpointing, for $MTBF_{0.9}$, $MTBF_{0.6}$ and different C-R parameters.

Similarly, Figure 5.2 shows the corresponding energy gains for large-scale deployments when xDVS governor is combined with checkpointing vs. an execution at nominal CPU settings and no checkpointing.

Our analysis shows that both RVSCap and xDVS mechanisms (and operation at reduced margins in general) can provide energy gains even at scale, and for rather costly checkpointing implementations. For instance, in a system with 1000 nodes, operation at reduced margins, through RVSCap, remains beneficial if $C + R$ is under 40 minutes for $MTBF_{0.6}$. For the much higher confidence level of $MTBF_{0.9}$, usage of RVSCap is beneficial as long as $C + R$ is less than 20 minutes. Similarly, in a system with 1000 nodes, usage of xDVS remains beneficial with energy gains at 17.5% and 6.7% for the confidence levels of $MTBF_{0.6}$ and $MTBF_{0.9}$, respectively, even for costly $C + R$ overhead at 40 minutes. xDVS provides more energy benefits compared with RVSCap because xDVS exploits the workload-dependent portion of the CPU voltage margins and can reduce further, according to the executing workload, the CPU operating voltage.

5.2 Hardware Mechanisms

The RVSCap mechanism, presented in Chapter 3, exploits the reduction of the static, workload-independent, portion of the CPU voltage margins to minimize the performance penalty induced for power-constrained execution. On the other hand, the xDVS mechanism, presented in Chapter 4, depends on the behavior of the current

workload, during a sampling period, as this is quantified by the monitored performance counters. Our experimental evaluation demonstrates that, for typical workloads, the performance counters can be successfully used as indicators to reduce the CPU supply voltage. However, the sampling of the performance counters and the supply voltage adjustment can be as frequent as every 10ms. The adaptivity frequency may be too low to capture events caused by specific – potentially malicious – workloads, namely voltage droop viruses, that produce large voltage fluctuations and expose the susceptibilities of the power delivery network of microprocessors [100, 101, 102]. Typically, such workloads consist of a periodic sequence of high activity and low activity instruction regions at a frequency equal to the resonance frequency of the power delivery network ($50-200MHz$ in modern CPUs). This pattern simulates the invocation of high CPU activity workloads immediately after a period of very low activity, which causes large di/dt swings and large voltage droops, as shown in prior work [103].

The problem of voltage droop detection and mitigation in modern CPU and GPUs is (and should be) addressed at the hardware level with specialized circuitry [104, 105]. A high-speed droop detection mechanism continuously monitors the power grid causing a rapid charge shunt to the V_{dd} rail to correct a voltage emergency within a few clock cycles. Ideally, these mechanisms could also inform the software stack when voltage emergencies are detected, acting as a trigger towards more conservative undervolting. To this end, the X-Gene 3 processor offers such voltage droop counters that can be monitored from the software [21].

In general, stronger error protection and error recovery mechanisms to correct timing violations are very helpful as extra protection during aggressive voltage scaling. Note also that voltage droop viruses are difficult to generate and may be different across not only different microarchitectures but also across different CPU parts. Moreover, on top of a realistic software stack, the background operating system activity (jitter) smooths out large voltage swings caused by such viruses [78]. To this end, we tried to produce large di/dt swings and large voltage droops by alternating phases of high and low activity at different frequencies, based on the findings of prior work [103]. We experimented on the Intel-based platforms, presented in the Chapter 2. However, our effort did not result in tighter CPU voltage margins, compared to those identified by our characterization. This is attributed to the fact that all the Intel systems we experimented on, in contrast with the X-Gene 2 processor used in prior work [103], feature a Load Line Calibration (LLC) mechanism to mitigate such steep voltage swings.

Chapter 6

A Framework for Large-Scale Experimentation at Reduced CPU Voltage Margins

In this Chapter we focus on the question "How to automate the complex experimentation process?". The work presented in Chapter 3 and Chapter 4 exploits the reduction of CPU voltage margins to improve the energy efficiency of program execution, and/or to minimize the performance penalty induced when the CPU operates under a power consumption limit. This effort involves:

- Characterization of the CPU voltage margins of different architectures and different parts of each architecture.
- Data collection for training the models used.
- Experimental evaluation of the techniques introduced in Chapters 3 and 4.
- Validation of the robustness of the techniques with long-running experiments, as discussed in Chapter 5.

All the aforementioned items mandate long, complex experimental campaigns, involving multiple systems and multiple workloads, with different computational characteristics. Moreover, this process needs to be reactive to errors potentially induced due to overly aggressive reduction of voltage margins (during the characterization phase) and able to recover and continue the experimental campaign. It also needs to be resilient to issues external to the experiments, such as power or network outages (especially in the case of long-running experimental campaigns, as is the case in the validation phase). The latter must be handled gracefully, so there is no information loss or wrong quantification of voltage margins.

We introduce $(\mathrm{XM})^2$ (eXtended Margins eXperiment Manager) which automates the evaluation of software on systems operating outside their nominal configuration

envelope. Although $(\text{XM})^2$ supports both bare-metal and OS-controlled execution, in the context of this dissertation, we will focus on the OS-controlled flavor.

6.1 Framework Objectives

Our vision is to provide a versatile and robust framework that facilitates the execution of such complex experimental campaigns on any target platform. To this end, the main objectives of $(\text{XM})^2$ are the following:

- To support platforms with different CPU architectures;
- To support operation at multiple CPU frequencies;
- To support multiple execution schemes of applications, such as single-, multi-instance, and multi-threaded;
- To detect any erratic system behavior due to voltage margins reduction;
- To recover the target platform in case of an error that caused the target platform to crash;
- To distinguish between errors due to operation at reduced voltage margins and external ones, such as power or network outages, and act accordingly;

6.2 $(\text{XM})^2$ for OS-controlled Execution

As Figure 6.1 shows, the core components of $(\text{XM})^2$ for OS-controlled execution are the Server and the Client components. More specifically, the Server component runs on a separate machine and is the orchestrator of the experimental campaigns. The Client component, which is responsible for servicing all the requests of the Server

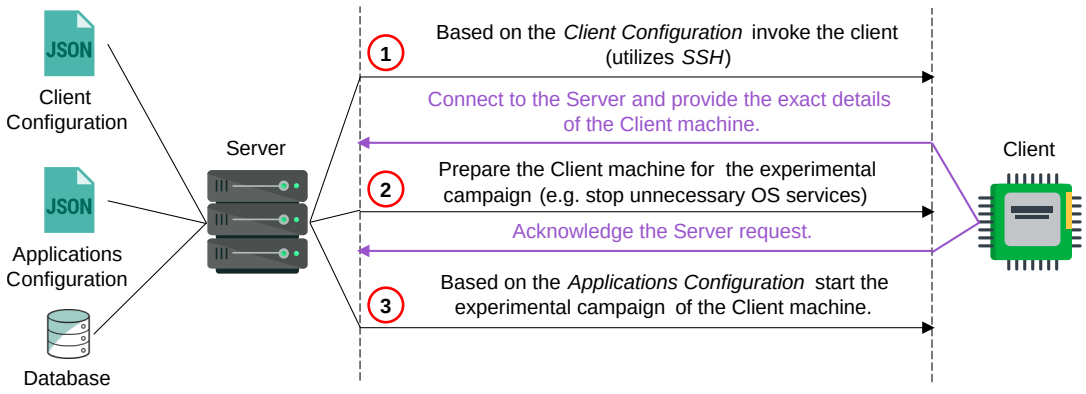


Figure 6.1: $(\text{XM})^2$ for OS-Controller execution overview.

component, runs on the target platform. Both of these components are written in Python, which is installed out-of-the-box in most Linux-based distributions. Python is the perfect match for the needs of $(\text{XM})^2$, as the language offers fast prototyping, a vast variety of packages, and, as it is a script-based language, it does not require any binary compilation to support different platform architectures. On the other hand, Python is an interpreted language which results in a larger memory footprint and lower performance when compared to a native binary. However, $(\text{XM})^2$ is by design I/O bound and, consequently, it neither induces measurable overhead, nor affects the observed behavior of the target system.

Initially, the Server component finds the IP of the orchestrator machine and binds a TCP port for the communication with the Client component. Afterward, the Server component, based on the *Client Configuration*, transfers the corresponding files and executes the Client component on the target platform, with the appropriate IP and TCP port arguments, by using the SSH protocol (through the *paramiko* pip package). Then the Client component sends an ACK payload to the Server. Later on, the Server, based on the Applications Configuration and the entities of the database, initiates the experimental campaign. The Client and Applications configuration, as well as the database functionality of $(\text{XM})^2$ will be presented in detail in the following Sections.

6.2.1 Client configuration

We need to provide the necessary information of the target platform, such as IP, credentials for SSH connection, etc., so that $(\text{XM})^2$ can remotely invoke the experimental campaigns. To this end, we designed a JSON-based configuration file, as shown in Listing 6.1, that provides all the necessary information to $(\text{XM})^2$. More specifically, based on the properties of *hostname*, *port*, *username*, and *password*, $(\text{XM})^2$ connects to the target platform and uploads all the necessary files of the Client component which are defined by the property *files* to the path as specified by the *dir* property.

After the files uploading, the Server component executes the Client component on the target platform and waits for an incoming connection to the corresponding TCP port. As soon as the Client sends the ACK payload, the Server initializes the target platform by sending the commands of the *init* property to the Client. Note that these commands are BASH shell commands. As shown in Listing 6.1, which is an example *Client configuration* for an Intel-based platform, at the initialization phase the Server disables the NMI watchdog of the Linux kernel and loads the MSR kernel module which is necessary for manipulating the operating frequency and voltage of Intel CPUs.

```
{
    "hostname": "10.64.16.50",
    "port": 22,
    "username": "*******",
    "password": "*******",
    "files": [
        "ClientMain.py",
        ...
    ],
    "dir": "/opt/characterizationClient",
    "init": [
        "echo 0 > /proc/sys/kernel/nmi_watchdog",
        "modprobe msr",
        ...
    ],
    "campaign_mode": "characterization",
    "frequency_nominal": "0x19e0001e1e",
    "frequency_cmd": [
        "wrmsr -a 0x774 %{frequency}"
    ],
    "undervolt_nominal": 0,
    "undervolt_cmd": [
        "/opt/intel_under %{undervolt}"
    ],
    "restart": {
        "type": "sftp",
        "username": "*******",
        "password": "*******",
        "hostname": "10.64.16.97",
        "cmds": [
            "restart_pc.py 10.64.16.50",
            "sleep 120"
        ]
    }
}
```

Listing 6.1: $(\mathrm{XM})^2$ Client Configuration file for Intel-based platforms.

The property of *campaign_mode* informs $(\mathrm{XM})^2$ on the kind of the experimental campaign. The possible values of this property, are *characterization* or *evaluation*. For the value of *characterization*, $(\mathrm{XM})^2$ will perform the characterization of CPU voltage margins, while for the value of *evaluation*, $(\mathrm{XM})^2$ will not control the operating points of CPU and will only evaluate, in terms of performance and (optionally) execution correctness, each application found in Applications configuration, that we will discuss in detail in the following Section. As a result, if we want to quantify the CPU voltage margins of a platform we set the value to *characterization*. When we need to evaluate the reliability of a mechanism that controls the CPU operating frequency and voltage, such as the two mechanisms presented in Chapter 3 and Chapter 4, we set the value to *evaluation*.

Another two important properties, that are needed by $(\mathrm{XM})^2$ when *campaign_mode* is set to *characterization*, are *frequency_nominal*, and *frequency_cmd* which, as the name implies, correspond to the frequency value under nominal CPU operation and the BASH command(s) by which the $(\mathrm{XM})^2$ can apply the target CPU frequency, respectively. In the same spirit, *undervolt_nominal* represents the nominal undervolt value, which for the Intel-based example is zero since underscaling the CPU operating

voltage is offset-based. *undervolt_cmd* provides the BASH command(s) by which the $(\mathrm{XM})^2$ can change the operating voltage of the CPU. Note that *frequency_cmd* and *undervolt_cmd* contain two special string tokens *%{frequency}* and *%{undervolt}*, respectively. These two tokens are filled accordingly, by $(\mathrm{XM})^2$ at runtime, depending on the stage of the characterization campaign.

Furthermore, as $(\mathrm{XM})^2$ evaluates software on systems operating outside their nominal configuration envelope, a way to reset the target platform is necessary, because an aggressive reduction of the CPU operating voltage will lead to a system crash. This is controlled by the property *restart*. In Listing 6.1, we configure the $(\mathrm{XM})^2$ to connect to a remote machine, through the SSH protocol (specified by the properties *type*, *username*, *password*, and *hostname*), and issue the BASH commands that will successfully reset the target platform. A detailed description of how we chose to implement this functionality will be presented in a later Section.

6.2.2 Applications configuration

At this point, $(\mathrm{XM})^2$ has initialized the target platform according to the *Client configuration* and the actual experimental campaign is ready to begin. However, $(\mathrm{XM})^2$ needs extra information about the applications. To this end, we designed a JSON-based configuration file, namely *Applications configuration*, which encapsulates all the necessary information for $(\mathrm{XM})^2$ to execute the corresponding applications. Listing 6.2 presents an example of such configuration that contains a single application, h264ref.

Initially, $(\mathrm{XM})^2$ parses each application, inside the configuration file, for the *frequencies* and the *cores* properties. According to these, $(\mathrm{XM})^2$ will create len(*frequencies*) * len(*cores*), in total, experimental sub-campaigns, for the corresponding application, that capture all the possible CPU cores utilization schemes for all the specified CPU operating frequencies. In case these properties are not specified for the application, $(\mathrm{XM})^2$ will create only one sub-campaign at the *frequency_nominal*, as specified in the Client configuration, and at all CPU cores utilization. Moreover, when *campaign_mode* is set to *evaluation*, $(\mathrm{XM})^2$ will not adjust any of the CPU operating points and it will create len(*cores*) sub-campaigns. The *name* property corresponds to the application name and it is the unique identifier of the application in the $(\mathrm{XM})^2$ database. $(\mathrm{XM})^2$ uses this identifier to distinguish between complete and pending sub-campaigns.

Another important property, when *campaign_mode* is set to *characterization*, is *undervolt_search_method*. This property defines how $(\mathrm{XM})^2$ will transition between

```
[
    {
        "name": "h264ref",
        "cores": [
            "0,1,2,3",
            "0,2"
            ...
        ],
        "frequencies": [
            "0x0808000808", // 800 MHz
            ...
            "0x1414001414", // 2000 MHz
            "0x1A1A001A1A", // 2600 MHz
            "0x2121002121"  // 3300 MHz
        ],
        "undervolt_search_method": "binary_half",
        "lower_bound": 150,
        "upper_bound": 400,
        "undervolt_step": 100,
        "preparation_cmds": [...],
        "pre_app_cmd": null,
        "app_cmd": "cd /opt/h264ref && taskset -c %{cpu_id} ./h264ref
            -d ./data/ref/input/foreman_ref_encoder_baseline.cfg >
            ./%{app_name}_%{cpu_id}",
        "post_app_cmd": null,
        "sdc_cmd": "cd /opt/h264ref && ./checkSDC.sh
            ./%{app_name}_%{cpu_id} ./golden",
        "sdc_pattern": "SDC detected",
        "multithreaded": false,
        "nominal_time": 600,
        "run_times": 3,
        "run_fixed": 1200
    }
]
```

Listing 6.2: $(\text{XM})^2$ example of Applications Configuration file that contains h264ref.

the different CPU voltage operating levels for a given sub-campaign and it can have the following values:

- **Linear**: In this search mode, $(\text{XM})^2$ decreases the CPU operating voltage linearly, starting from the *lower_bound* undervolting level with a step defined by the *undervolt_step* property, until erratic system behavior or a crash are detected. This search mode is recommended for target platforms that exhibit narrow voltage margins, such as the X-Gene processors presented in Table 2.1.

- **binary_half**: As shown in Listing 6.3, in this search mode, $(\text{XM})^2$ initially performs the *Linear* search mode, with a voltage reduction step of 100mV, until erratic system behavior or a crash are detected. The resulting undervolting level will be the *upper_bound*. Also, during the search of *upper_bound*, each voltage reduction step that finishes without any erratic system behavior or crashes becomes the *lower_bound*. When a crash is detected, the search algorithm knows that the voltage reduction that corresponds to the V_{min} (does not result to erratic system or application behavior) lies between *lower_bound* and *upper_bound*. Therefore, it recursively performs a binary search between those two bounds, in order to accurately identify the respective undervolting level. This search

```
upper_bound = None
lower_bound = 0
undervolting_level = 100

# linear search
while upper_bound is None:
    status = characterize(undervolting_level)
    if (status == OK):
        lower_bound = undervolting_level
    else:
        upper_bound = undervolting_level
        continue

    undervolting_level += 100

# binary search
while upper_bound - lower_bound >= 5:
    undervolting_level = lower_bound + (upper_bound - lower_bound)/2
    status = characterize(undervolting_level)
    if (status == OK):
        lower_bound = undervolting_level
    else:
        upper_bound = undervolting_level

vmin = lower_bound
```

Listing 6.3: pseudocode of *binary_half* search mode of $(\mathrm{XM})^2$.

mode is recommended for target platforms that exhibit wide voltage margins, such as the Intel-based processors presented in Table 2.1.

The *preparation_cmds* define all the necessary initialization BASH shell commands that an application may require. $(\mathrm{XM})^2$ executes these commands only once at the beginning of each sub-campaign. In the same spirit, the properties *pre_app_cmd* and *post_app_cmd* include shell commands that are executed every time before and after the actual application execution, respectively. These two properties are complementary and they serve potential needs such as monitoring the system while the application is running. As an example, we could utilize these two properties to start Linux perf and measure performance counters of the CPU just before the execution and right after the termination of the application.

The property *app_cmd* is the shell command that $(\mathrm{XM})^2$ will issue to execute the application. Moreover, based on the boolean *multithreaded* property, $(\mathrm{XM})^2$ identifies the application as multi-threaded or single-instance. In case the value of *multithreaded* property is true, $(\mathrm{XM})^2$ will search for the special string *%{cpu_id}* in *app_cmd* and it will replace it, at runtime, with the corresponding CPU cores count of the sub-campaign. Also, $(\mathrm{XM})^2$ will set the CPU affinity of the application threads to the corresponding CPU cores, by wrapping the *app_cmd* with the proper taskset system command. On the other hand, when the value of *multithreaded* property is false, $(\mathrm{XM})^2$ will issue as many *app_cmd* instances as the CPU cores count of the sub-campaign, and it will set, for each one independently, the proper CPU affinity through taskset system command.

If the properties *sdc_cmd* and *sdc_pattern* are defined, after the application execution finishes, $(\mathrm{XM})^2$ invokes the shell commands of *sdc_cmd* and captures the stdout of the process. If the regular expression pattern defined in *sdc_pattern* matches the stdout, then $(\mathrm{XM})^2$ considers this undervolting level not safe and proceeds to evaluating another undervolting level, according to the *undervolt_search_method*. To this end, it is up to the programmer to supply an executable that can detect if a Silent Data Corruption (SDC) occurred and there are any unexpected differences in the application output. Another critical property, that $(\mathrm{XM})^2$ utilizes to classify an undervolting level as safe or not, is the *nominal_time* property. Based on this property, $(\mathrm{XM})^2$ determines if an application is taking significantly longer than it would have if the CPU was operating at a nominal voltage point. If this property is not defined in the configuration, then $(\mathrm{XM})^2$, before initiating the application sub-campaigns, extracts and saves this information by executing once the application at nominal CPU operating voltage. Moreover, $(\mathrm{XM})^2$ monitors the message buffer of the kernel for Machine Check Exception errors (MCE). In case there is an error reported, $(\mathrm{XM})^2$ will mark this undervolting level as not safe and continue accordingly the characterization sub-campaign.

Two important properties that $(\mathrm{XM})^2$ relies on and control the end of a sub-campaign when none of the errors, such as SDCs, MCEs or exceeding of execution time are detected, are *run_times* and *run_fixed* properties. The *run_times* property represents the number of times, in a sub-campaign, that an application has to run without any error so $(\mathrm{XM})^2$ can consider the corresponding undervolting level as safe. The *run_fixed* property provides the same functionality but $(\mathrm{XM})^2$ considers the undervolting level as safe when the duration of consecutive application executions surpasses the time specified. When both properties are defined, $(\mathrm{XM})^2$ terminates the campaign upon one of the two becoming true.

Moreover, since $(\mathrm{XM})^2$ also supports *evaluation* experimental campaigns, we introduce two command line arguments used for this kind of experimental campaigns, namely *–repeat* and *–time*. More specifically, *–repeat* argument represents the number of times that $(\mathrm{XM})^2$ will execute all the applications in the Application Configuration to deem the campaign as completed. Similarly, *–time* represents the duration in minutes of the campaign that $(\mathrm{XM})^2$ will keep executing all the applications in the Application Configuration. In case both command line arguments are missing for an *evaluation* experimental campaigns, $(\mathrm{XM})^2$ will finish the campaign when all applications in the Application Configuration are executed one time. Table 6.1 summarizes the key configuration differences which control whether $(\mathrm{XM})^2$ will be instructed to perform either a *characterization* or an *evaluation* experimental campaign.

Table 6.1: $(\mathrm{XM})^2$ configuration differences to perform a *characterization* or an *evaluation* experimental campaign.

Campaign Type	*Characterization*	*Evaluation*
Command Line Arguments		–time 33120
Client Configuration	campaign_mode = characterization	campaign_mode = evaluation
		init = include BASH command to invoke the mechanism under evaluation

6.2.3 Database

A characterization campaign of CPU voltage margins or an evaluation of execution correctness for a mechanism, that controls the CPU operating points, can take a significant amount of time. As a consequence, for both cases, it must be able to recover from errors that are not due to the reduction of voltage margins, such as network or power outages. In this spirit, we designed $(\mathrm{XM})^2$ to be robust. More specifically, $(\mathrm{XM})^2$ utilizes the *blitz-DB* python package and keeps track of every step of the experimental campaign. That said, $(\mathrm{XM})^2$ detects a network outage, through the Server component, by pinging continuously the Cloudflare DNS 1.1.1.1, during an experimental campaign. If the DNS is unreachable, $(\mathrm{XM})^2$ pauses the experimental campaign until the reachability is restored. On the other hand, if the Server component can reach the DNS but cannot communicate with the Client component this is classified as an error due to operation at reduced voltage margins. Furthermore, for the detection of a power outage, we assume that it will also affect the Server component of $(\mathrm{XM})^2$ and the experimental campaign will resume, from the last recorded step in the database, when power is restored.

6.3 Node Resetting Controller

As already mentioned, the main objective of $(\mathrm{XM})^2$ is the evaluation of systems operating outside their nominal configuration envelope, hence, it needs a way of recovering the target platform in case of a crash caused by an aggressive voltage margins reduction. To satisfy this requirement we designed a circuit, shown in Figure 6.2, that can be used to restart the target platform, when a Baseboard Management Controller (BMC) is missing. More specifically, the circuit acts as a switch that controls two optocouplers. We choose to use optocouplers, as they offer an extra layer of protection, in terms of galvanic circuit isolation, since they do not require a common ground connection with the target platform. When it is powered on, the optocouplers allow current to flow simulating a closed circuit. In this way, the optocouplers can replace the actual physical switch that powers on or off the target platform. Although using

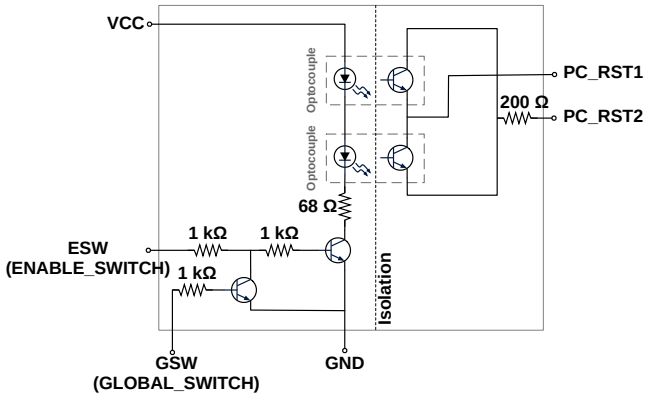


Figure 6.2: Complementary circuit for resetting the target platform.

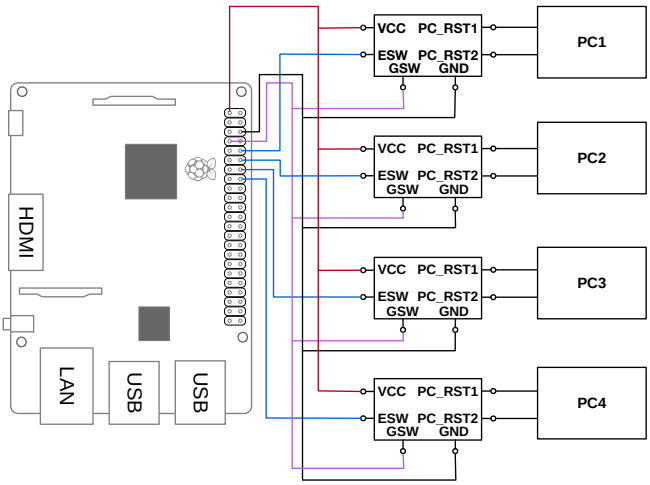


Figure 6.3: Controlling the state of multiple platforms with a Raspberry Pi SoC.

only one optocoupler could achieve the same results, we opt to use two to prevent any damage to the target platform in case the physical switch requires a particular polarity. The aforementioned design is combined with an SoC that allows the programming of I/O pins (in our case a Raspberry Pi board (Figure 6.3)). Therefore, $(\text{XM})^2$ can connect to the SoC and command a power-cycling, thus recovering the target platform. This is the deployment scheme we used in both the characterization and the evaluation campaigns, presented in Chapter 3 and Chapter 4.

6.4 Related Work

Several approaches target to analyze and exploit hardware operation under faults. In particular, [100] presents a new class of fault attacks that target the software-exposed energy management mechanisms present in modern commodity devices, such as in ARM/Android devices. They exploit the voltage and frequency scaling capabilities of modern processors to compromise system security, by injecting faults during code

execution and extracting cryptographic keys from the ARM TrustZone. In contrast to our work, which focus on analyzing and understanding how and under what circumstances modern CPU microarchitectures fail when executing code at unsafe configurations, this work focuses on the security risks raised by modern energy management techniques.

Several research efforts focus on introducing different fault-injection techniques. As an example, [106, 107] introduce deterministic and reproducible fault injection techniques at the pin-level of a processor to validate and propose hardware fault-tolerance mechanisms. Moreover, [108, 109] implement a software-level fault injection, which models complex systems with great accuracy, however, ensuring that the simulated models are realistic and , at the same time, controlling simulation time are significant challenges. The authors in [110, 111, 112] introduce fault injection simulators that provide high accuracy in both the location and the timing of the fault, however they introduce significant overhead. [113] enables fault injection in the back-end of the LLVM compiler. Furthermore, [114] presents LLFI, a fault injector that, also, works at the LLVM compiler's intermediate representation (IR) level of the application and studies the effect of different injection choices on their resilience, namely instruction type, register target and number of bits flipped. In contrast to these works, $(\mathrm{XM})^2$ can execute software natively on the targeted system, at subnominal operating points, therefore it provides native execution time and does not rely on any fault models and assumptions.

Finally, the authors in [115] present an automated framework to support system-level voltage and frequency scaling characterization that supports the APM X-Gene 2 platform. Even though $(\mathrm{XM})^2$ can be used to extract the same results, our work supports various processor platforms, different execution modes, such as OS-controlled and bare-metal execution, evaluation of mechanisms that control CPU operating points and the collection of profiling data, based on which the behavior of the target platform can be further analyzed.

Chapter 7

The Individual and Combined Energy Efficiency Effects of Approximate & Heterogeneous Computing

In this Chapter, we answer the questions “Can approximate and heterogeneous computing be combined?” and “What is the effect in the energy efficiency vs. quality of results tradeoff?”. As we discussed earlier, it is pretty common for programmers to treat all parts of the code as equally important, without considering the contribution of each part to the quality of output, and the Quality of Service (QoS) requirements of the application. Programmers can, however, introduce approximations in less significant parts of their code in an educated manner, in order to improve the energy efficiency of code execution. As a result, approximate computing opens up new possibilities for power/energy management and energy footprint minimization of applications.

Another software specialization approach, that yields energy savings of code execution, is heterogeneous computing. Heterogeneous computing combines different architectures, each appropriate for specific computational patterns, within the same system. Consequently, programmers can selectively execute parts of their code on the appropriate compute units, such as a GPU that performs embarrassingly parallel computations efficiently, and significantly improve the energy efficiency of the application.

Motivated by the energy savings of approximate and heterogeneous computing, we investigate whether their combination can yield favorable solutions in the energy efficiency vs. quality of results trade-off. We developed 7 applications, that are part of AcHEe (Approximate Computing and Heterogeneity for Energy efficiency) benchmark suite [116], using mainly OpenCL nomenclature (further details are provided in Section 7.1.2). Therefore they can target any architecture and accelerator device supporting OpenCL. Also, for each application, we provide both accurate and approximate implementations of its computationally intensive parts. The latter exploit

different types of approximations, carefully chosen to balance between energy efficiency and quality degradation of results. We evaluate the behavior of the applications on two different platforms (CPU and GPU), under different degrees of approximation and we quantify both the isolated and the cumulative effect of heterogeneity and approximations to energy footprint and quality of results. Our set of applications can be used to assess the combined effect of approximations and heterogeneity on different, existing and emerging accelerators, as well as, act as a test-bed for evaluating and comparing different approximation techniques.

The main contributions and outcomes of our work are the following:

1. To the best of our knowledge, this is the first work that investigates the combination of Heterogeneous and Approximate computing and evaluates the trade-off between quality of results and energy efficiency.

2. We experiment with both large-scale and kernel-scale applications that allow us to evaluate more realistically the impact of heterogeneity and approximate computing on the energy footprint of applications.

7.1 Platform Assumptions

7.1.1 Hardware assumptions

The applications we experiment with assume a heterogeneous system: apart from one or more CPUs in the host, accelerators may also be available for executing computational kernels. To this end, we have experimented using GPUs as accelerators. Moreover, we assume different address spaces between the host and accelerators.

Our methodology is not based on the capability to dynamically set voltage and frequency of compute devices (DVFS) and the experimental evaluation, presented in Section 7.3 does not exploit DVFS . DVFS is rather orthogonal to approximate and heterogeneous computing from the perspective of hardware, therefore the evaluation explicitly focuses on quantifying the energy gains that one can achieve with heterogeneity and approximation and the quality/energy trade-offs enabled by the latter.

7.1.2 Software assumptions

Applications are written using OpenCL as the main programming model. More specifically, we express computational kernels – both accurate and approximate – in OpenCL, as this facilitates code reuse and portability among the different execution devices of a heterogeneous system. Therefore, we assume the OpenCL compiler and runtime support for all target architectures.

We use a set of OpenMP-like extensions, presented in Section 2.2, on top of OpenCL to eliminate common OpenCL development overheads (such as queues creation and manipulation, kernel compilation and enqueue, data management, scheduling, etc) and to allow the controlled execution of either the accurate or the approximate version of each kernel[1]. More specifically, for each computational task, the programmer, using #pragma directives, may provide an approximate OpenCL implementation (in addition to the accurate one). The degree of approximation (approximation level) is controlled using a $ratio$, a single "analog" knob that specifies the minimum percentage of tasks that will be executed accurately. A compiler, based on LLVM [117] lowers #pragma directives to calls to an underlying runtime system. The end-user can specify the desired level of approximation by setting the *ratio* parameter in the [0-100] range. The ratio corresponds to the minimum percentage of tasks that will be executed accurately. Therefore, with limited programmer effort and no intrusion to the rest of the code, multiple approximation techniques can be implemented and evaluated.

Finally, we assume that hardware monitors power consumption and performance, and offers an API to programmatically access this information from the runtime. This is true for most modern computational devices.

7.2 Applications

In this Section, we provide an overview of the 7 applications and we briefly discuss the essence of the approximation techniques we applied. The reader can access the code of the benchmarks for additional implementation details [116].

Table 7.1 summarizes the applications and the respective application domains. For each application, the first two columns provide information on algorithmic characteristics, namely whether the application is iterative – in the sense of performing

[1] In any case, all benchmarks can be expressed in vanilla OpenCL with some additional programmer effort

Table 7.1: List of target applications and their characteristics.

Program	Domain	Iterative	Input Dependent	Approximation Type	Quality Metric
LULESH	Physics simulation	✗	✗	Drop	Relative error
MD	Physics simulation	✗	✗	Drop	Relative error
Monte Carlo	PDE solver	✗	✗	Drop	Relative error
K-Means	Vector quantization	✓	✓	Lower accuracy	Relative error
Fisheye	Image Processing	✗	✗	Drop	PSNR
DCT MV	Video transcoding	✗	✓	Drop	PSNR
SPS-Stereo	Computer vision	✗	✓	Sync. relaxation	PSNR

steps to converge to a solution, not including applications which perform time steps – and whether its behavior is input – instead of input size-dependent. The last two columns provide the approximation technique(s) and the quality metric used for the evaluation of results.

7.2.1 LULESH

LULESH [118] is a hydrodynamics physics simulation. It solves the Sedov blast wave problem for a liquid in the 3D space, exploiting Lagrangian hydrocodes. Liquids in hydrodynamics exhibit *boundary conditions* [119] allowing the partitioning of the original problem domain into further sub-domains. Due to symmetry, LULESH solves the problem in one sub-domain which is the one octant of the initial domain. This sub-domain is defined as the computational region and the problem is solved using staggered mesh approximation [120]. Therefore the mesh is defined as a structure of cubes where the edges are known as *nodes* and their centroids as *elements*. The thermodynamic variables such as energy and pressure are calculated at the elements, while kinematic variables such as position and velocity are calculated at nodes. LULESH initializes all the variables to *zero*, except for the position of the nodes and the volume of the elements. Also, the blast takes place at the center of the initial domain, so it sets an initial value to the energy of the element that lies closest to the blast.

As it is common in hydrodynamics simulations, the problem is solved in discrete time steps (Lagrange steps). Each step calculates, at first, the time increment Δt_n needed in order to advance the values of the variables at the new time t_{n+1} corresponding to $t_{n+1} = t_n + \Delta t_n$. Then it constructs a force at each mesh node, taking into consideration the hourglass filter contribution and calculates the acceleration, velocity and the new position of the nodes. These variables are needed to update the element variables and to calculate the artificial viscosity, pressure, and energy, according to the material model properties. Finally, it recalculates the time constraints that are needed for the given Δt_n of the next time step.

Approximation: We introduce an approximate version where we eliminate the calculation of the hourglass force for some elements. The contribution of this force is needed for the elements that are close to the blast, but we can drop this calculation for those that are further away. More specifically, for the fully approximate execution (ratio=0.0) we calculate the corresponding contribution of each particle to the hourglass force with such a cut-off distance. On the other hand, for a fully accurate execution (ratio=1.0) every particle contributes to the hourglass force. For the in-between approximation ratios, the cut-off distance is used partially for some particles.

Moreover, we move simulation faster in time compared with the accurate version, by increasing the Δt_n more rapidly at each time step. This is a valid approximation, because LULESH exhibits *energy conservation*. More specifically, we observe that the energy conservation law (the total energy of an isolated system remains constant) is not affected by this kind of approximation and the relative error of results remains low, compared with the accurate execution.

7.2.2 Molecular Dynamics

Molecular Dynamics (MD) methods are widely used to simulate particle/atoms systems ranging from solids, liquids to the motion of stars and galaxies. For our benchmarking needs, we simulate a system of N liquid Argon atoms, which interact under inter-atoms force and in the absence of external forces. The governing law of such a system is the equation of motion for particles, which is the numerically integrated *Newton's* equation of motion. The interacting forces among atoms stem from Lennard-Jones potential [121]. Although this is not the most faithful representation of the potential energy surface, it is widely used due to its computational simplicity. Lennard-Jones encapsulates a mildly attractive force for large distances and a strongly repulsive for very short distances. The constant motion of particles in liquids causes particles to collide with the bounding box and with each other. This results in a short deformation of their electron clouds, known as *dipole moment* [122] and particles attracted to each other in the form of dipoles. The repulsive force originates from the inability of any atom to diffuse through another. Lennard-Jones force is described by the following equation:

$$F(r) = -\frac{dV(r)}{dr} = \frac{24\epsilon}{\sigma}\left[2\left(\frac{\sigma}{r}\right)^{13} - \left(\frac{\sigma}{r}\right)^{7}\right] \tag{7.1}$$

where V is the intermolecular potential between the two atoms or molecules, ϵ is the well depth (material dependent), σ is the distance at which the intermolecular potential between the two particles is zero (material dependent), r is the distance between both particles.

In the absence of external forces, the system is in equilibrium state and its *energy* is *conserved*. In this state, static properties such as temperature and pressure are measured as averages over time, and *energy conservation* can be monitored by measuring the total energy of the system periodically throughout the simulation.

Approximation: We approximate the simulation by setting a cut-off distance. This distance defines a surrounding region of a particle; interactions are computed only among particles within the region. This approximation is realistic, as the dominant

term in Equation 7.1 is negligible for large distances. After experimentation, we found that an acceptable distance, enabling energy conservation is 40 Å.

7.2.3 Monte Carlo PDE solver

Monte Carlo methods are widely used in statistical simulations. Here, we focus on stochastic numerical solvers for deterministic elliptic Partial Differential Equation (PDE) problems. As previous work has shown [123, 124], Monte Carlo methods are the only realistic choice for solving several difficult non-linear problems and remain a good choice for linear problems such as particle diffusion.

This Monte Carlo implementation [125] allows the solution of a PDE into a sub-domain D of the original domain Ω. To do that, one needs to calculate the values at the interface Γ of D and Ω, obviously without solving the PDE in Ω. The method performs a, potentially large, number of random walks on spheres for each point of Γ. A random walk starts from a point of Γ and stops when reaching the external boundary of Ω. Each walk produces an estimation of the value at the respective start point. The method combines the estimations from all random walks for each point and then calculates an interpolant. This can be done in parallel for all points of Γ. Then the PDE can be solved inside D only. The aforementioned approach can lead to less costly solutions for specific regions of interest of large PDE problems.

Approximation: An approximation methodology to all Monte Carlo methods is to drop a percentage of the random experiments (random walks in our case) and their corresponding computations. An approximate, lightweight methodology is also used to decide how far from the current location the next step of a random walk should move. More specifically, we substitute double-precision variables with single-precision ones and we utilize the native, fast math implementation of trigonometric formulas (such as cosine and sine) provided by each accelerator vendor. Fast math versions of mathematic operations are more performant, approximate (non IEEE 754 compliant) implementations of the corresponding accurate but costly computations.

7.2.4 K-Means

K-means is an iterative algorithm for grouping data points in a multi-dimensional space into k clusters. It is used extensively in data-mining and machine learning. The number of dimensions contained within the aforementioned space is the length of features of input points. K-Means comprises two phases. In the first phase, points are assigned to OpenCL threads, which independently: (a) first identify the nearest cluster using Euclidean distance as a metric, and then (b) assign the point to the selected cluster. In the second phase, *K-means* computes the centroid of the points in

each cluster to serve as the new cluster center. This process is repeated until no points migrate between clusters during an iteration (convergence), or after a user-specified number of iterations.

Approximation: In K-Means, computing the cluster centroid correctly is important, as it is much harder to recover from a wrong cluster centroid, which will erroneously attract new data points in the cluster in the following iterations. Consequently, centroid calculations are always executed accurately. However, the assignment of individual points to clusters is error-tolerant. Even if a point is misclassified into the wrong cluster, it typically does not significantly affect the centroid. Moreover, it will usually be classified into the correct cluster during a subsequent iteration of the algorithm. To this end, we use a simpler version of the Euclidean distance and we consider only half of the total dimensions, to assign the points to the appropriate clusters.

7.2.5 Fisheye

Fisheye [126] is an image or video processing application, which transforms images captured through an ultra-wide-angle (fisheye) lens to the perspective space. Fisheye got its name after Snell's window, which is how a fish sees from beneath the water. Specifically, as a fish vision has hemispherical geometry, so does the captured image. This means there is a deviation from rectilinear projection, which is called barrel distortion. This radial distortion allows scenes –to be captured– to map around a sphere, facilitating the capturing of an ultra-wide scene into a finite image area.

The code first associates pixels of a rectilinear projection, perspective space image, to points (potentially between captured pixels) in the original distorted image (inverse mapping). The value of each output pixel is calculated by performing bi-cubic interpolation on the values of a 4×4 window of neighboring pixels around the corresponding point in the distorted image. Finally, a low pass filter can be applied to the output image to remove potential artifacts.

Approximation: The approximate version performs the inverse mapping step accurately. Although this step is computationally intensive, it is executed only once for each lens and requested field of view. Approximation focuses on bi-cubic interpolation, which has to be performed for each image frame. Instead of calculating the value of output pixels by interpolating the 4×4 neighborhood of the corresponding point in the input, the approximate algorithm simply uses the value of the nearest neighboring pixel.

7.2.6 DCT MV

DCT MV is an abstract skeleton of a video encoder. More specifically it combines two of the most essential steps of a video encoding, *Discrete Cosine Transform* (DCT) [127] and residual *Motion Vector* (MV) [128] extraction. Sophisticated video compression is necessary to meet the requirements of network bandwidth, quality, and performance. In video sequences, usually, the only difference between several adjacent frames is the relative motion of some areas, produced either by a moving camera or object. This means that the majority of information that represents a new frame remains the same. Video encoders identify the differences between frames and express them as motion vectors of frame-blocks. Afterward, they apply DCT compression. Therefore, the decoder can reconstruct multiple frames using a single reference frame and limited additional information.

DCT MV first partitions the frames of a video into batches of sequential frames. The first frame of each batch is considered as the reference frame for the remaining frames in the batch. The residual motion vectors are extracted for blocks of 8×8 pixels. Afterwards, a difference image is produced between each frame and the reference frame, based on the corresponding motion vector information. Finally, a quantized DCT is applied to each difference image. For quality checking purposes, after the above process, a decoding phase reconstructs the encoded frames.

Approximation: The approximate version performs motion vector extraction within a reduced search window down to 8×8 pixels from 128×128 of accurate execution. Even if the motion recognition fails for a few blocks, the quality and compression loss will be minimal due to the small block size. Another approximation has been applied in the calculation of quantized DCT coefficients. Particularly, we exclude the calculation of the half downright DCT coefficients of each block, corresponding to high spatial frequencies. Such frequencies carry, in most images, very low energy. Moreover, the human eye is less sensitive to high frequencies.

7.2.7 SPS-Stereo

SPS-Stereo (SPS) is a stereo vision application producing a dense disparity map of a static scene captured with a stereo pair camera. It combines a Stereo Global Matching (SGM) algorithm [129] and a slanted plane algorithm which assumes that the 3D scene is piece-wise planar and the motion is rigid or piece-wise rigid [130, 131]. Robust and dense computation of depth information is necessary for many machine vision applications, such as driver assistance systems and robotics, and is a cheap, yet effective alternative to pricey RGB-D cameras.

SGM is the most compute-intensive and time-consuming part. It considers pairs of images with known intrinsic and extrinsic orientation (stereo-camera characteristics) and is responsible for an initial disparity image estimation. Disparity estimation is the task of identifying the projection point of the same 3D real-world point in two or more images taken from distinct viewpoints. The most challenging task of a disparity estimation algorithm is to identify correctly a point and its different references from different viewpoints in the same 3D world, despite the high level of ambiguity.

SGM first emphasizes edges (capped sobel) and maps the intensity values of the pixels within a square window to a bit string (census transform) for both base and match images. These bit strings are then used to produce an initial estimation of pixel costs for each image pair. Pixel costs quantify the similarity of pixel regions between base and match images. Later on, the final pixel costs are calculated by aggregating different pixel costs from different scanlines and redistributing the new pixel costs appropriately, by a predefined penalty factor. Finally, SGM multiplexes different scanlines then extracts the global minimum pixel cost and matches it with the corresponding disparity reference for each pixel. This step consists of a forward and backward analysis of image pairs. The results of the two passes are added and produce the final disparity estimation.

Approximation: Finding the global minimum of pixel costs suffers from sequential dependencies across consecutive rows of the image. The original implementation exploits SIMD instructions but the level of parallelism of SIMD prohibits an efficient OpenCL implementation. For the accurate execution, we settle for a CPU-friendly implementation that uses the OpenCL vectorization extensions. The approximate version deliberately relaxes these synchronization constraints (barriers). We expect a loss of quality of objects disparity in the captured scene in exchange for extra parallelism. Consequently, in the final disparity dense estimation, one local minimum emerges for each scanline.

7.3 Evaluation

The experimental evaluation was carried out on a dual-socket system equipped with two Intel Xeon E5 2695 processors, clocked at 2.3 GHz, with 128 GB DRAM and an NVIDIA Tesla K80 GPU. The operating system is Ubuntu 14.04 server, using the 3.16 Linux kernel. All applications (Table 7.1) utilize one GPU chip of the K80, therefore the combination of two CPUs results in more power consumption. To calculate energy consumption on GPU, we measure the power drain with the interval of 2 ms using NVML [132] and multiply the average by the execution time. On CPU,

we utilize the Running Average Power Limit (RAPL [133]) counters to measure the energy consumption on each socket.

For each application, we provide a spider plot with 5 axes. Applications are executed on CPU and GPU at different degrees of approximation (corresponding to different lines in the plot). We report the impact of approximations on energy efficiency for CPU and GPU w.r.t. the accurate execution on the same device (*Energy Improv % (CPU)* and *Energy Improv % (GPU)* axis respectively) and on quality w.r.t. the accurate execution (*Quality % (w.r.t. accur.)* axis). Moreover, for each approximation level, we report the energy gain on the GPU w.r.t. the CPU version at the same approximation level (*Energy Gain % (GPU w.r.t. CPU)*), to identify cases where there are device-specific variations in the energy efficiency between the accurate vs. the approximate version. The last axis (*Energy Gain % (GPU w.r.t. CPU accur.)*) reports the energy gain of the GPU approximate execution (again at different degrees of approximation) w.r.t. the accurate execution of the CPU. This metric quantifies the combined energy gains of both heterogeneity and approximate computing over a conventional execution.

As a general observation, we find that in most applications it is possible to improve the energy efficiency both by exploiting heterogeneity, and by using approximate computing at the expense of a controlled reduction in the quality of results. Particularly, in the following spider plots, we expect a gradual quality loss analogous to the energy efficiency gain.

7.3.1 LULESH

We evaluate LULESH for a staggered mesh of $1,000,000$ cubes. As a quality metric, we use the relative error of total system energy between accurate and approximate executions.

Initially, as shown in Figure 7.1, we observe that the accurate execution on GPU, exploiting only heterogeneity of hardware, results to 86.4 % less energy consumption than the accurate execution on CPU. LULESH exhibits a high degree of parallelism and fully utilizes the computational power of the GPU. Moving to a fully approximate execution (ratio = 0.00), the relative error is 1.3 %. Elements that are far from the Sedov blast do not significantly affect the spread of the blast wave, hence ignoring them does not introduce significant error. Also, the energy conservation property of the system minimizes the error when we increase Δt_n, thus forwarding the simulation more rapidly. The total energy gains from the approximations reach up to 43.6 % and 41.6 % on the GPU and CPU, for the fully approximate execution, respectively.

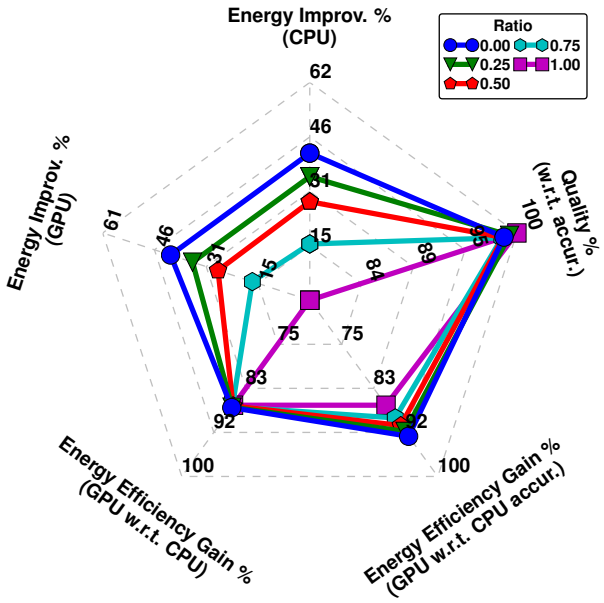


Figure 7.1: LULESH evaluation

These energy savings can be mainly attributed to the the increased simulation time-step approximation. Furthermore, when we combine heterogeneous computing with approximations (execution on GPU with ratio = 0.00) we observe a significant 93.7% improvement in energy efficiency w.r.t. the fully accurate execution on CPU.

7.3.2 Molecular Dynamics

We evaluated MD for 32768 particles confined to a $600 \times 600 \times 600$ box (Å^3), for a total of 1000 time steps, corresponding to 1 fs each. For quality quantification we used the relative error of average total energy of the system –sampled every 100 time-steps– between the accurate and approximate execution.

Figure 7.2 shows that exploiting heterogeneous computing (executing on GPU) for MD, results in 89.1 % energy efficiency improvement w.r.t. fully accurate execution on CPU. For the approximate version, quality is virtually unaffected (0.2 % rel. error). Because of the high number of particles, the cut-off distance for interactions respects homogeneity of particle distribution, resulting in the negligible error of the kinematic properties of the system and thereby of its energy. More specifically, Figure 7.3 shows that despite applying a cut-off distance, the spatial distribution of the particles between the approximate (particles with red color) and the accurate execution (particles with blue color) remains similar. Another important observation is that approximations result in higher energy gains on CPU than GPU (e.g. for ratio = 0.00: 72.8 % and 39.2 % for CPU and GPU respectively). The main reason leading to this behavior is the better energy efficiency of the GPU due to its higher

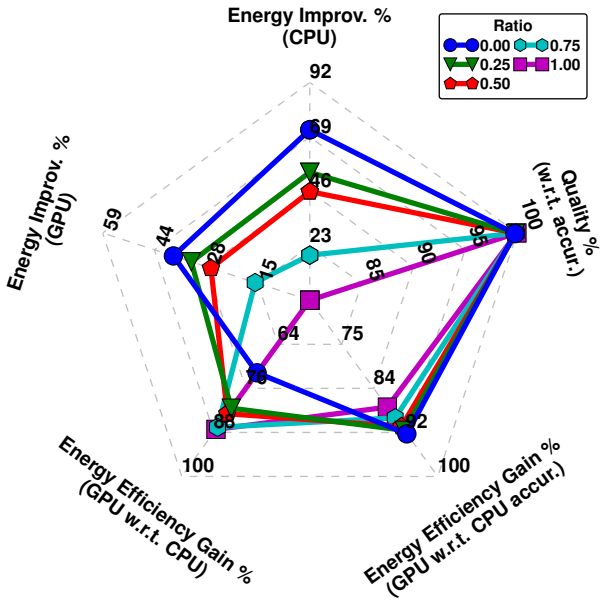


Figure 7.2: MD evaluation

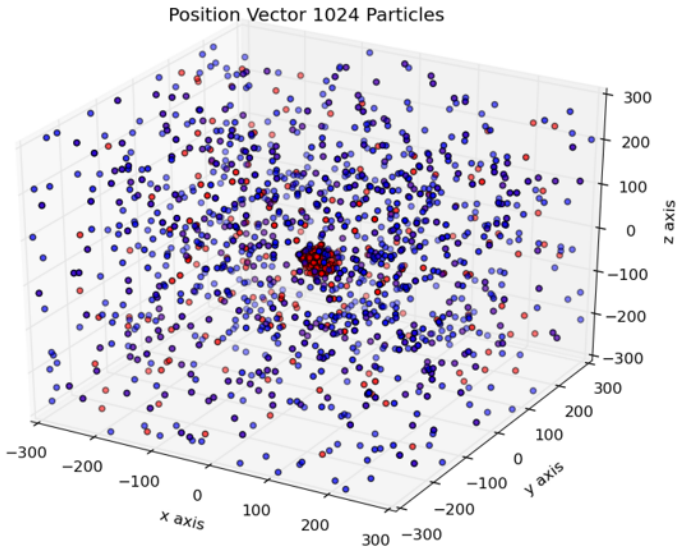


Figure 7.3: Positions of particles for a fully approximate (red) and accurate (blue) execution).

level of parallelism compared with the CPU. When combining approximate with heterogeneous computing, energy consumption is further reduced by 92.1% w.r.t. fully accurate execution on CPU.

7.3.3 Monte Carlo PDE solver

For the Monte Carlo PDE solver, we estimated the interface values of a 150×50 points subdomain within a larger 2D PDE domain. For each point, we perform 50

random walks. Quality is quantified by the relative error of estimated interface values by the approximate vs the accurate Monte Carlo execution.

As shown in Figure 7.4, fully accurate execution on GPU, for Monte Carlo, results in 84.3% reduction of energy consumption w.r.t. fully accurate execution on CPU. Moreover, the fully approximate version improves energy efficiency by 79.6 % and 47.3 % for GPU and CPU, respectively, at the expense of 0.01% relative error. The very limited extent of quality loss is expected because of the stochastic and redundant nature of Monte Carlo methods in general. Finally, we observe that fully approximate execution on the GPU (ratio 0.00) reduces the energy footprint of Monte Carlo by 96.8%.

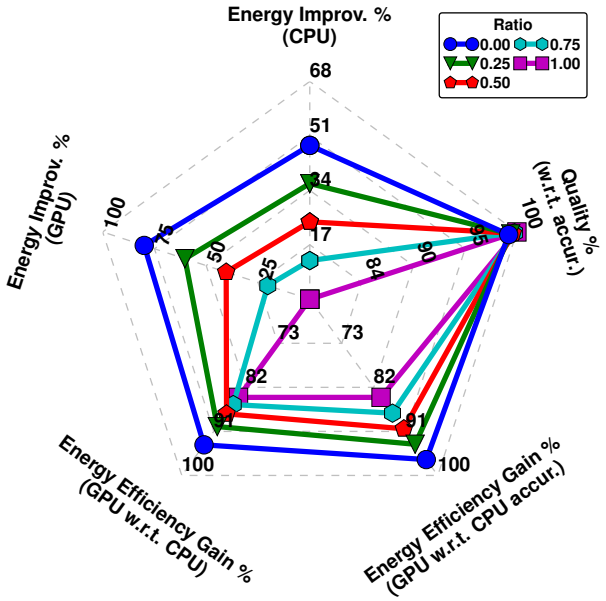


Figure 7.4: Monte Carlo PDE solver evaluation

7.3.4 K-Means

In K-Means we use a 59 MB subset of the KDD Cup 1999 Data input set [134] and try to classify data objects in 35 discrete clusters. We measure the quality using the Relative Mean Square Error (RMSE) between the points of the cluster and its centroid. We report the relative error of RMSE between accurate and approximate executions.

Figure 7.5 shows that accurate execution on GPU results in 75.5% reduction of energy consumption w.r.t. accurate execution on CPU. Moving to approximate computing, classifying data objects taking into account only part of their features yields similar results, in terms of quality, with accurate execution. More specifically, the relative error of the misclassified points for the fully approximate execution (ratio =

0.00) is 0.01 %. However, K-Means has an interesting behavior not visible in Figure 7.5. Approximating the classification of points to clusters results in points moving across clusters more frequently than in the accurate versions. This, in turn, tends to increase the workload of the (always accurate) computation of new cluster centroids. Still, the energy consumption is reduced by 38.1% and 45.6% for CPU and GPU fully approximate executions, respectively. Furthermore, the fully approximate execution on GPU reduces the energy consumption by 84.6% w.r.t. fully accurate execution on CPU.

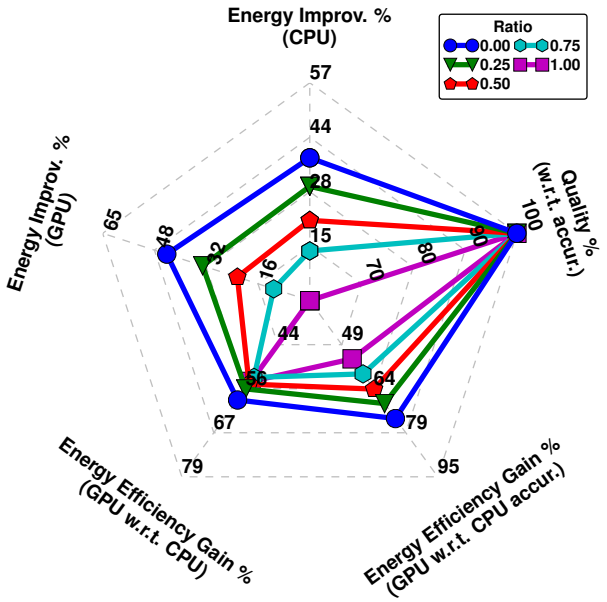


Figure 7.5: K-Means evaluation

7.3.5 Fisheye

For Fisheye we used high-resolution images (2592×1944) and had the application repeat the visual distortion correction procedure 10 times to artificially increase execution time. Although the accurate execution result has an infinite PSNR, for illustration purposes, we limit PSNR to 58 dB.

As shown in Figure 7.6, execution on GPU benefits from the implemented level of parallelism of Fisheye. Energy consumption is 92.8 % lower compared to CPU accurate execution. For CPU and GPU the energy gain between approximate and accurate execution is 12.3 % and 23.1 %, respectively, even though we approximate only one (bi-cubic interpolation) out of three kernels in the application (the other two being inverse mapping and the final low pass filter). The approximate version results to a very high PSNR, at 48 dB. An image with a PSNR of 48 dB is excellent for

most applications. Combining execution on GPU with approximations, the energy footprint of Fisheye is reduced by 94.1% w.r.t. fully accurate execution on CPU.

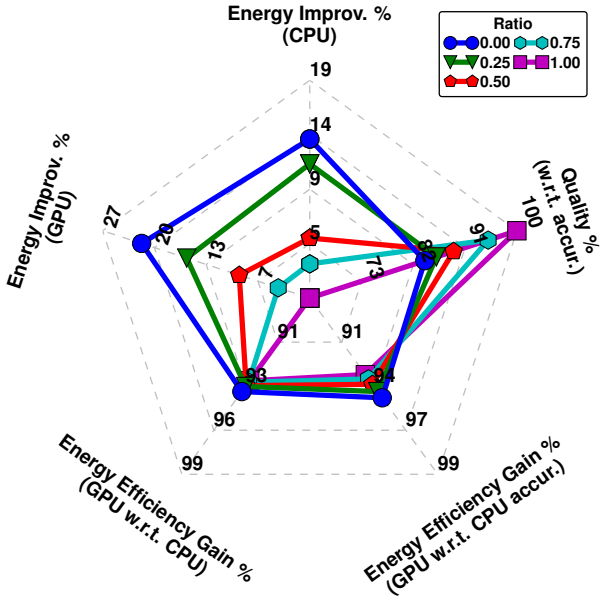


Figure 7.6: Fisheye evaluation

7.3.6 DCT MV

In DCT MV we used as input 10 frames of video, containing footage of road traffic, with pixel resolution 1352×512. The video contains fast-moving objects, therefore it is appropriate for evaluating the effect of approximations on output quality. For each frame, we calculated the PSNR of the reconstructed frame, using the original one as golden input. The quality metric is the PSNR of the worst quality frame in the sequence (the lowest PSNR of the sequence). Quantized DCT is a lossy compression algorithm, therefore even during accurate execution reconstructed frames do not have infinite PSNR. The PSNR of the accurate version is $38\ dB$ which corresponds to 100 % of the spider plot.

Figure 7.7 shows that execution on the GPU results in higher energy efficiency by 61.6 % for the accurate version compared with the execution on CPU, because of the high level of exploitable parallelism of the application. Moreover, DCT MV quality reaches a minimum PSNR of 33 dB in fully approximate execution. The fully approximate execution has ~ 95 % less energy consumption, w.r.t. fully accurate execution, on both CPU and GPU. This is expected, as half of the DCT coefficients are not calculated and the motion detection window is only 8×8 (resulting in ~ 99.6 % fewer computations). The quality is still very high, as shown in Figure 7.8; the PSNR is ~ 5 dB lower than that of the fully accurate execution. The compression rate is

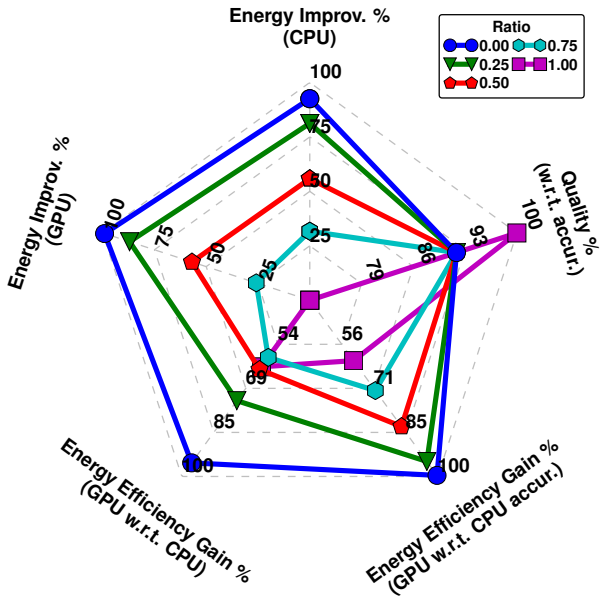


Figure 7.7: DCT MV evaluation

$3.58\times$ and $2.97\times$ for the fully accurate and fully approximate versions respectively. Once again, this is expected because fully accurate execution introduces more zeros and smaller numbers in motion vector residuals, and by extension to DCT coefficients, which in turn that leads to better compression rate for fully accurate execution

(a)

(b)

Figure 7.8: (a) DCT MV fully accurate output, (b) DCT MV fully approximate output

in contrast with fully approximate. Another observation is that different levels of approximate execution result to almost the same energy efficiency gains on both CPU and GPU (e.g. for fully approximate execution gains of 92.5 % and 99.1 % for CPU and GPU respectively). However, combining execution on GPU with approximations reduces the energy consumption of DCT MV by 98.7% w.r.t. to the fully accurate execution on CPU.

7.3.7 SPS-Stereo

We evaluated SPS using the *KITTI Vision dataset* [135] and Peak Signal to Noise Ratio (PSNR) to quantify quality. In reality, the PSNR of the accurate execution is infinite, however for illustration purposes, we limit it to 50 dB. As the implemented approximation prohibits execution on CPU, the approximate version runs entirely on GPU. On the contrary, the accurate version combines both CPU and GPU. Therefore, we supply a spider plot with differentiated axes from the rest of the applications. One axis corresponds the energy improvement for the combined energy consumption of GPU and CPU, for different approximation levels. A second one corresponds to output quality, similarly to other applications. The remaining three axes of the spider plot are not used.

As we observe in Figure 7.9, the fully approximate execution (ratio = 0.00) results in a PSNR of 37.269 dB, which is excellent for most applications. Figure 7.10e depicts the different between the output of fully accurate execution (Figure 7.10c) vs the one of fully approximate execution (Figure 7.10d). Calculating local minimums

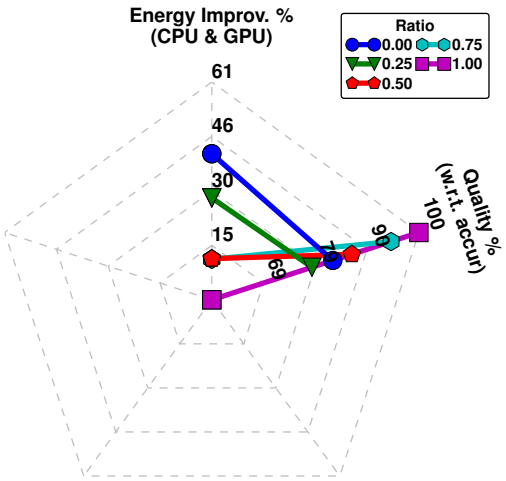


Figure 7.9: SPS-Stereo evaluation

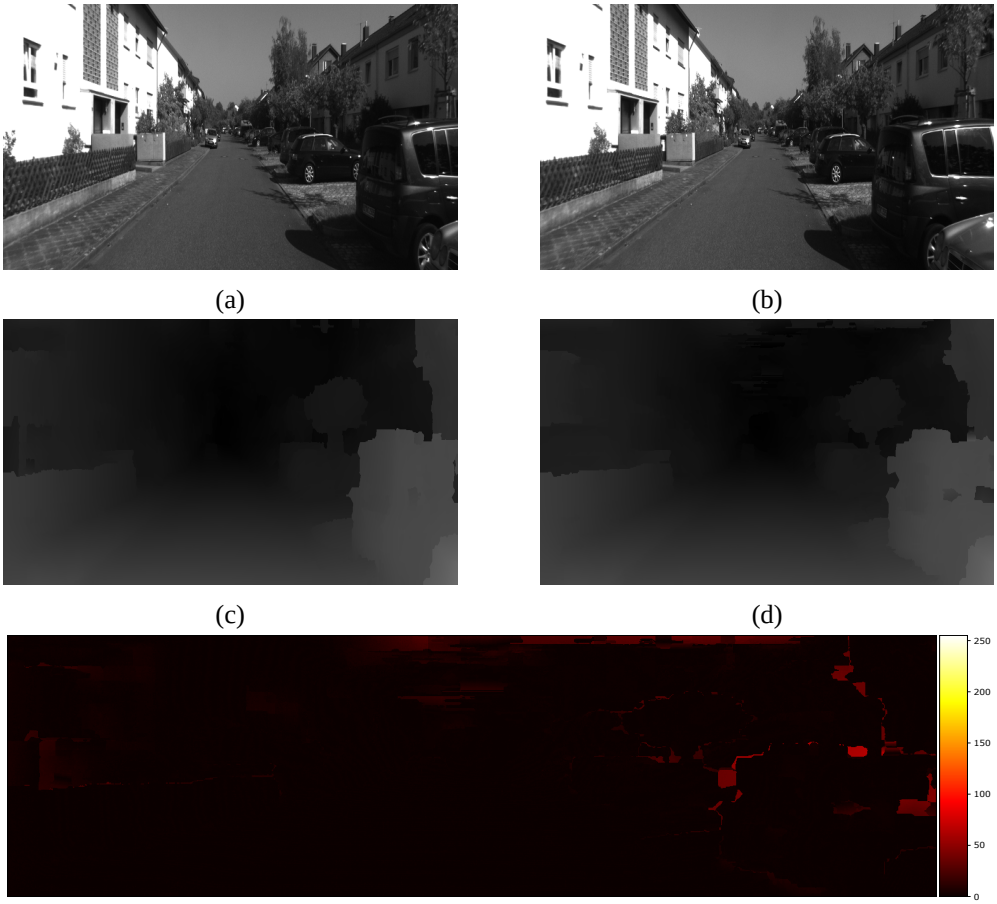


Figure 7.10: (a) (b) Respectively, left and right images of the captured scene (c) SPS-Stereo fully accurate output, (d) SPS-Stereo fully approximate output, (e) Heatmap of pixel intensity differences of fully accurate vs fully approximate output.

of pixel costs in each scanline instead of the global minimum leads to the manifestation of small artifacts. It also reduces the fidelity of the outlines of objects in the scene. However, we observe that object positioning and disparity in the scene (which are the main outputs of the algorithm) are still correct. In terms of energy efficiency, the fully approximate execution results in 44.8% less energy consumption compared with the fully accurate execution.

7.4 Related Work

To the best of our knowledge, this work is the first that investigates the impact of combining heterogeneous and approximate computing, at configurable levels of approximation, on the trade-off between energy efficiency and quality of output. Our work differentiates from existing studies and benchmark suites in several ways.

AxBench [136] offers separate CUDA and C/C++ applications to study, in isolation, the impact of approximations on CPU and GPU. It exploits two generic approximation techniques, Neural Processing Units [137] and loop perforation [138]. Both techniques are application-agnostic and require extensive profiling/training. In contrast, we target multiple architectures from a single OpenCL implementation and we exploit programmer wisdom to employ application-specific approximations. Therefore, our quantification is not affected by the effectiveness of generic approximation techniques.

ApproxIt [139] is an approximate computing framework, which adjusts approximation modes dynamically to minimize computational intensity. In contrast with our work, ApproxIt can only be applied to iterative applications and does not consider heterogeneity.

EnerJ [140] defines a data-centric approximate type system. It uses type qualifiers to distinguish between precise and approximate computations statically, letting explicit control of the approximate data flow to the programmer. Approximate computations are mapped to low-power operations and storage, isolated from the precise components. Our task-centric approach, given a quality constraint (ratio), can decide the approximation level dynamically at task granularity, without isolating accurate and approximate data. Also, we consider the heterogeneity features of the underlying system.

OpenDwarfs [141] is a heterogeneous benchmark suite written in OpenCL. Its code is architecture-agnostic. Shoc [142], also written in OpenCL, focuses on comparing the features and performance of modern parallel architectures using microkernels. Moreover, it compares OpenCL and CUDA implementations. The Rodinia [134] benchmark suite, in its latest version (3.1), offers an OpenCL implementation alongside CUDA and OpenMP, for most of the applications. It covers a variety of patterns, common among parallel applications. It aims to facilitate the study of heterogeneous systems. In contrast to those three benchmark suites, which are performance-oriented, we evaluate our applications in terms of energy efficiency. Furthermore, we also evaluate the cumulative potential of approximate- on top of heterogeneous computing.

Contrary to all aforementioned benchmark suites, we experiment with both large-scale and kernel-scale applications that allow us to evaluate more realistically the impact of heterogeneity and approximate computing on the energy footprint of applications. Such a level of realism is not possible with small-scale software modules and kernels only.

Chapter 8

Concluding Remarks

8.1 Summary

In the previous Chapters, we presented our work on exploiting intrinsic pessimistic hardware guardbands, hardware- and software-heterogeneity to improve the energy efficiency of computing systems. We began by identifying the core challenges of exploiting intrinsic hardware guardbands and software heterogeneity and afterwards, we introduced our solutions to minimize the energy footprint of computing systems.

The very first challenge is to address the question "How to exploit the voltage margins reduction?". To this end, we presented the characterization results of the voltage margins for four different, commercially available, processors. Based on our findings, we observed that the reduction of voltage margins can be distinguished into a static and a – non-negligible on certain architectures – dynamic, workload-dependent, part. More specifically, for processors that exhibit wide voltage margins, the extent the CPU operating voltage can be reduced without leading to imminent system crashes, depends on the executing workload. On the other hand, for processors with significantly lower margins, the degree of voltage reduction is almost constant, especially for scenarios where all CPU cores are utilized.

To turn these opportunities into actual energy savings we designed, developed and evaluated – on real, commercially available ARM- and Intel-based systems – two mechanisms that exploit the CPU voltage margins. The first mechanism, Reduced Voltage Scaling power Capping (RVSCap), exploits the reduction of the static CPU voltage margins to minimize frequency throttling when executing on power-constrained environments. The second mechanism, Extended Dynamic Voltage Scaling (xDVS) which is an online voltage scaling governor, exploits the workload-dependent CPU voltage margins and reduces, according to the computational characteristics of the executing workload, the supply voltage of modern multi-core Intel x86-64 CPUs.

Given that CPU operation at reduced voltage margins may potentially limit the effectiveness of a reliability safeguard introduced by manufacturers, we validated the

robustness of these mechanisms with a set of long-running experiments, on multiple systems, under the control of RVSCap and xDVS, without observing any erratic behavior. Moreover, we showed that significant energy gains can be achieved even when taking into account the extra cost (due to the potentially lower MTBF) of checkpointing and recovery in large-scale systems.

Furthermore, we designed and implemented eXtended Margins eXperiment Manager ($(\mathrm{XM})^2$), a framework that deals with the complexity of the experimental campaigns and, thus, largely simplifies and automates the experimental setup to identify CPU voltage margins, to extract data used to train models, to evaluate RVSCap and xDVS and to validate the aforementioned mechanisms. $(\mathrm{XM})^2$ supports multiple systems, multiple configurations, and numerous different workloads.

Finally, we investigated whether the combination of heterogeneous and approximate computing can yield favorable solutions in the energy efficiency vs. quality of results tradeoff. More specifically, under the premise that not all parts or execution phases of a program affect the quality of its output equally, we modified 7 applications to exploit both heterogeneity and approximations. To this end, we programmed the applications using mainly OpenCL nomenclature, so they can target any architecture and accelerator device supporting OpenCL, and for each application, we provide both accurate and approximate implementations of its computationally intensive parts. Also, we evaluate them on heterogeneous platforms (comprising CPUs and GPUs) and quantify the isolated and combined effect of heterogeneous and approximate computing.

8.2 Conclusions

In the context of this Dissertation, we show that exploiting intrinsic hardware guardbands is both possible and can result in significant energy savings for four different commercial CPUs, the Intel x86-64 Xeon E3 and Haswell i7 x86-64, as well as the ARMv8 64-bit AppliedMicro X-Gene 2 and X-Gene 3.

Initially, we experimentally show that CPU voltage margins reduction has a significant impact on the performance during power-constrained execution. We observe that for restrictive power caps, CPU power capping with reduced voltage margins (RVSCap) can improve the performance by 64% for Xeon E3 (RAPL), 30% for X-Gene 2 (DFSCap) and 34% for X-Gene 3 (DFSCap), on average. Also for less restrictive power caps, it is possible to meet the same performance with the conventional capping mechanisms, however with reduced node power footprint, on average by 24%, 4% and 5% lower for Xeon E3 (RAPL), X-Gene 2 (DFSCap) and X-Gene 3

(DFSCap), respectively. Also, CPU voltage margins reduction leads to greater performance gains even when compared with the state-of-the-art solution in CPU power capping (PUPiL) by 24%, 22% and 5% on average for Xeon E3, X-Gene 2 and X-Gene 3, respectively. Noticeably, we show that the synergy between these is not only possible but also outperforms every other approach, in terms of performance.

Furthermore, we found that the extent of voltage margins varies across different CPU microarchitectures, different parts of the same microarchitecture, different levels of CPU core utilization and – most importantly – on some architectures (Intel-based) can be wide and workload-dependent. In this context, we showed the importance of using a diverse set of single- and multi-instance/threaded benchmarks for CPU voltage margins characterization. We also found – and experimentally proved for Intel's Skylake and Haswell microarchitectures – that it is possible to train models that predict V_{min}, at execution time, for the workload at hand. Those models are based on information from hardware performance counters which quantify characteristics of software/hardware interaction. We show – by designing and implementing xDVS – that those model predictions are exploitable to dynamically vary CPU voltage at runtime. The experimental evaluation of xDVS on real hardware, executing a wide range of benchmarks and larger-scale applications, showed that significant energy savings are possible, by up to 42.68% and 34.37% over the standard Intel *P-state* DVFS governor for parts of the Skylake and Haswell families, respectively.

CPU operation at reduced voltage margins may be beneficial, in terms of energy efficiency, however, it might potentially result to lower system reliability. A long experimental validation indicates that our mechanisms can be used to operate systems in sub-nominal CPU voltages, without compromising system reliability. Still, we statistically (and pessimistically) estimate the potential effect on the MTBF of Xeon E3 CPUs, when operating with reduced voltage margins. Our exercise proved that the exploitation of CPU voltage margins can result in energy gains, even at larger scales, and even when considering the increased checkpointing overhead, assuming a potentially reduced system MTBF.

However, we claim that hardware support for the detection and mitigation of voltage emergencies is useful to safeguard against aggressive (and potentially malicious) codes, such as voltage droop viruses. Voltage emergencies should not be treated differently from other emergencies (such as memory access bound violations, execution of illegal instructions, use of illegal operands, etc.) which can undermine the safe operation of a computing system, and are traditionally expected to be identified by hardware.

Finally, in the context of this dissertation, we exploit hardware and software heterogeneity to improve the energy efficiency of computing systems. To this end, we

introduced a set of 7 non-trivial applications written to exploit both heterogeneity and approximations for energy footprint minimization and we find that it is possible to combine heterogeneity and approximations to aggressively improve the energy efficiency of applications. Also, we show that taking into account domain expert wisdom allows these savings to occur, in most cases, without uncontrolled degradation of output quality.

8.3 Future Work

Our work on exploiting the intrinsic CPU voltage guardbands opens several research opportunities for future work.

All the mechanisms, presented in the context of this Dissertation, as well as, prior works that exploit the CPU voltage margins for the same platforms as the ones we use, rely on an excessive characterization process. It would be worthwhile to investigate whether there are proxies (symptoms) that signify that CPU voltage reduction enters a critical zone in terms of CPU operation reliability. Identifying such proxies would be highly beneficial for online mechanisms that exploit CPU voltage margins reduction. Such mechanisms would become more responsive and safe, without having to potentially sacrifice opportunities for greater energy savings or to rely on complex prediction models for the appropriate voltage margins reduction.

Another interesting direction for future work would be to study the synergy of operation at reduced voltage margins with approximate and heterogeneous computing. Such a combination presents an ideal opportunity to drastically minimize the energy footprint of an application. On the one hand, as we discussed earlier, different microarchitectures have different extents of exploitable voltage guardbands. Heterogeneous systems integrate multiple architectures within a single computing system. In the context of this Dissertation we studied the voltage margins of CPUs and focused on their exploitation. Accelerator devices, such as GPUs, FPGAs and NPUs could be the focus of future work. On the other hand, as we already discussed in this dissertation, in many cases the exploitable extent of voltage margins is workload-dependent. Since approximate computing results in changes to the instruction-mix of an application, a further investigation of how these changes affect the voltage margins reduction is necessary. Such work would add another dimension (that of operating voltage reduction) to the problem of mapping computations to components of a heterogeneous system, and the already complex and interesting tradeoff among power, performance, and quality of results.

Appendix A

Related Publications

[1] Panos Koutsovasilis, Konstantinos Parasyris, Christos D Antonopoulos, Nikolaos Bellas, Spyros Lalis. Dynamic Undervolting to Improve Energy Efficiency on Multicore X86 CPUs. *IEEE Transactions on Parallel and Distributed Systems*, TPDS, *(Under review. Submitted August 2nd 2018, major revision requested May 5th 2019, revised version submitted July 14th 2019.)*

[2] Panos Koutsovasilis, Christos D Antonopoulos, Nikolaos Bellas, Spyros Lalis, George Papadimitriou, Athanasios Chatzidimitriou, Dimitris Gizopoulos. Impact of CPU Voltage Margins on Power-Constrained Execution. *IEEE Transactions on Sustainable Computing* , TS-USC, *(Under revision. Submitted December 5th 2019, major revision requested February 17th 2020.)*

[3] Christos Kalogirou, Panos Koutsovasilis, Christos D Antonopoulos, Nikolaos Bellas, Spyros Lalis. Exploiting CPU Voltage Margins to Increase the Profit of Cloud Infrastructure Providers. In *Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, (CCGRID)

[4] Konstantinos Parasyris, Panos Koutsovasilis, Christos D Antonopoulos, Nikolaos Bellas, Spyros Lalis. A framework for evaluating software on reduced margins hardware. In *Proceedings of 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* , (DSN)

[5] Georgios Karakonstantis, Konstantinos Tovletoglou, Lev Mukhanov, Hans Vandierendonck, Dimitrios S Nikolopoulos, Peter Lawthers, Panos Koutsovasilis, Manolis Maroudas, Christos D Antonopoulos, Christos Kalogirou, Nikos Bellas, Spyros Lalis, Srikumar Venugopal, Arnau Prat-Perez, Alejandro Lampropoulos, Marios Kleanthous, Andreas Diavastos, Zacharias Hadjidimitriou, Pangiota Nikolaou, Yannakis Sazeides, Pedro Trancoso, George Papadimitriou, Manolis Kaliorakis, Athanasios Chatzidimitriou, Dimitris Gizopoulos, Shidharta Das. An energy-efficient and error-resilient server ecosystem exceeding conservative

scaling limits. In *2018 Design, Automation & Test in Europe Conference & Exhibition*, (DATE)

[6] Christos Kalogirou, Panos Koutsovasilis, Manolis Maroudas, Christos D Antonopoulos, Spyros Lalis, Nikolaos Bellas. Edge and Cloud Provider Cost Minimization by Exploiting Extended Voltage and Frequency Margins. In *Proceedings of 2017 International Conference on Parallel Computing (PARCO)*, "Advances in Parallel Computing", Vol.32, pp. 814-823, 9/2017, ISBN 978-1-61499-842-6 (print). IOS Press

[7] Panos Koutsovasilis, Christos Kalogirou, Christos Konstantas, Manolis Maroudas, Michalis Spyrou, Christos D Antonopoulos. AcHEe: Evaluating approximate computing and heterogeneity for energy efficiency. In *Parallel Computing 73 (2018)* (pp. 52-67)

Appendix B

Contribution to Joint Publications

This appendix discusses my contribution to each of the publications, presented in Appendix A.

In [1], I designed a methodology to exploit the CPU voltage margins reduction depending on the computational characteristics of the executing workload. Initially, to suit the needs of voltage margins quantification, I reverse engineered the API to underscale the operating voltage for Intel CPUs, since at the time this was only available for the Windows operating system, through Intel's XTU application. Afterwards, for six Intel-based systems, I extracted the characterization results of the CPU voltage margins, for multiple applications. Motivated by the variability, of the characterization results, I investigated with Dr. Parasyris Konstantinos the feasibility of monitoring performance counters and predicting the minimum CPU supply voltage at which the reliability of CPU operation is not compromised. To this end, I introduced support of several extra counters for Intel-based systems to the Linux perf tools (CPU Voltage ID, CPU thermal counters, etc.). I profiled all applications, under investigation, collecting all performance counters available on Intel-based systems. Dr. Parasyris Konstantinos created a script that tested all the available fitting models, provided in the sklearn python library, that took as input my profiling data and predicted the appropriate voltage margins reduction. However, the transition from just a prediction model to an online governor that manipulates the CPU operating points is not trivial. More specifically, such an online governor has to be robust, detect changes in the CPU utilization topology and adjust the CPU operating points without compromising the reliability of system operation. Having these attributes in mind, I designed, deployed and evaluated, on the six systems under investigation, the xDVS governor which utilizes the resulting prediction model with the highest predictive value, continuously adjusts the supply voltage and drives the CPU to a more energy-efficient, yet safe, zone of operation.

In [2], I investigated the impact of CPU voltage margins reduction on power-constrained execution. More specifically, motivated by prior research in CPU power

capping, which minimizes the performance penalty induced in power-constrained execution, but lacks the exploiting of intrinsic hardware guardbands, I evaluated operation at reduced voltage margins for various existing power capping mechanisms, such as RAPL. That said, to enable a comparison for X-Gene processors, which lack a CPU power capping mechanism, I developed DFSCap, a software-based CPU power capping mechanism that controls only the CPU frequency. Moreover, I produced the characterization results for the Intel-based systems. The respective characterization for the the X-Gene processors was performed by Dr. Papadimitriou George. Also, Dr. Papadimitriou George provided the micro-viruses benchmarks used – among other benchmarks – during the characterization process of the systems. Motivated by the findings of this investigation, I designed, implemented and evaluated RVSCap, a power capping mechanism, that supports multiple platforms and exploits CPU voltage margins reduction to minimize the performance penalty induced under power-constrained execution. Noticeably, my mechanism leads to greater performance gains even when compared with the state-of-the-art solution but, most importantly, the synergy between these is not only possible but also outperforms every other approach.

Additionally, for both [1] and [2], I conducted a long-running (23 days) experimental evaluation of both xDVS and RVSCap, on 16 Skylake systems, and I provided a pessimistic lower-bound of the systems MTBF, due to operation at reduced voltage margins. Based on this MTBF and the findings of prior work that quantifies the overhead of fault-tolerant mechanisms, I performed a statistical analysis that estimates the respective energy gains of xDVS and RVSCap for larger-scale deployments where a fault-tolerance mechanism, such as checkpointing and restore, is necessary, even for operation at nominal points.

In [3] we exploit CPU voltage margins to increase the profit of cloud infrastructure providers. More specifically, we introduce data center execution coordination and node configuration policies that optimize the profit of the cloud infrastructure provider taking into account SLAs and energy consumption. Since this is a simulation-based study, I contributed models are realistic and based on measurements on actual off-the-shelf systems. To this end, I developed two prediction models, one for the node power consumption and, another, for the expected performance degradation w.r.t. frequency underscaling. The power consumption model can predict the consumption at the plug of the node for CPU operation at both reduced margins and nominal points. Also, the performance degradation model estimates the impact of frequency underscaling based on the profiling data of multiple applications on actual hardware.

In [4], we introduce $(\mathrm{XM})^2$ a framework, co-designed by Dr. Parasyris Konstantinos and myself, that automates the experimental campaigns for the evaluation

of software on systems operating outside their nominal configuration envelope. I implemented the OS-controlled execution mode of $(\text{XM})^2$, which is compatible with multiple platforms, reactive to errors potentially induced due to overly aggressive reduction of voltage margins, able to recover and continue the experimental campaign, and resilient to issues external to the experiments (such as power or network outages).

In [7], I modified 6 real-world applications to exploit heterogeneous systems (execution on GPUs). Moreover, for all 6 applications, I designed and implemented approximations that further minimize the energy footprint of the execution, without significantly impacting the quality of results. For LULESH, I contributed to the design and evaluation of approximations. The actual implementation of LULESH was performed by Mr. Kalogirou Christos.

Bibliography

[1] *OECD*. https://www.oecd.org/sti/ieconomy/data-driven-innovation.htm. Accessed: 2020-1-27.

[2] Georgios Karakonstantis et al. "An Energy-Efficient and Error-Resilient Server Ecosystem Exceeding Conservative Scaling Limits". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 1099–1104.

[3] Jonathan Koomey. "Growth in Data Center Electricity Use 2005 to 2010". In: *A report by Analytical Press, completed at the request of The New York Times* 9 (2011).

[4] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. "Data Center Energy Consumption Modeling: A Survey". In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 732–794.

[5] Shidhartha Das et al. "A Self-Tuning DVS Processor Using Delay-Error Detection and Correction". In: *Solid-State Circuits, IEEE Journal of* 41.4 (2006).

[6] John L Henning. "SPEC CPU2006 Benchmark Descriptions". In: *ACM SIGARCH Computer Architecture News* 34.4 (2006), pp. 1–17.

[7] Cloyce D Spradling. "SPEC CPU2006 Benchmark Tools". In: *ACM SIGARCH Computer Architecture News* 35.1 (2007), pp. 130–134.

[8] Jack Doweck et al. "Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake". In: *IEEE Micro* 37.2 (2017), pp. 52–62.

[9] Edward A Burton et al. "FIVR—Fully Integrated Voltage Regulators on 4th Generation Intel® Core ™ SoCs". In: *In Proceedings of Twenty-Ninth Annual international conference on Applied Power Electronics Conference and Exposition (APEC)*. IEEE. 2014, pp. 432–439.

[10] Howard David et al. "RAPL: Memory Power Estimation and Capping". In: *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*. ACM. 2010, pp. 189–194.

[11] Huazhe Zhang and Henry Hoffmann. “Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid techniques”. In: *ACM SIGARCH Computer Architecture News* 44.2 (2016), pp. 545–559.

[12] AppliedMicro (APM). *APM883832-X3 | X-Gene 3© Multi-Core 64-bit Processor*. https://en.wikichip.org/w/images/2/22/832-X3_PB.pdf. 2016.

[13] Michalis Spyrou et al. “Energy Minimization on Heterogeneous Systems through Approximate Computing”. In: *Mini-Symposium on Energy and Resilience in Parallel Programming*. Edinburgh, Scotland, 2015.

[14] Aaftab Munshi et al. “The OpenCL Specification”. In: *Khronos OpenCL Working Group* 1 (2009), pp. l1–15.

[15] Michael Ferdman et al. “Clearing the Clouds: a Study of Emerging Scale-out Workloads on Modern Hardware”. In: *ACM SIGPLAN Notices*. Vol. 47. 4. ACM. 2012, pp. 37–48.

[16] Christian Bienia et al. “The PARSEC Benchmark Suite: Characterization and Architectural Implications”. In: *Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT)*. ACM. 2008, pp. 72–81.

[17] Jack J Dongarra, Piotr Luszczek, and Antoine Petitet. “The LINPACK Benchmark: Past, Present and Future”. In: *Concurrency and Computation: practice and experience* 15.9 (2003), pp. 803–820.

[18] George Woltman. *Prime95*. 2012. url: https://www.mersenne.org/download/.

[19] Daniel Hackenberg et al. “Introducing FIRESTARTER: A Processor Stress Test Utility”. In: *2013 International Green Computing Conference Proceedings*. IEEE. 2013, pp. 1–9.

[20] Konstantinos Parasyris et al. “A Framework for Evaluating Software on Reduced Margins Hardware”. In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2018, pp. 330–337.

[21] George Papadimitriou, Athanasios Chatzidimitriou, and Dimitris Gizopoulos. “Adaptive Voltage/Frequency Scaling and Core Allocation for Balanced Energy and Performance on Multicore CPUs”. In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2019, pp. 133–146.

[22] G. Papadimitriou et al. “Harnessing Voltage Margins for Energy Efficiency in Multicore CPUs”. In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, (MICRO)*. 2017, pp. 503–516.

[23] Konstantinos Tovletoglou et al. “Measuring and Exploiting Guardbands of Server-Grade ARMv8 CPU Cores and DRAMs”. In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018. doi: 10.1109/dsn-w.2018.00013.

[24] G. Papadimitriou et al. “Micro-Viruses for Fast System-Level Voltage Margins Characterization in Multicore CPUs”. In: *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2018, pp. 54–63. doi: 10.1109/ISPASS.2018.00014.

[25] E. Cai and D. Marculescu. “TEI-Turbo: Temperature Effect Inversion-Aware Turbo Boost for Finfet-Based Multi-Core Systems”. In: *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2015, pp. 500–507. doi: 10.1109/ICCAD.2015.7372611.

[26] W. Lee et al. “Dynamic Thermal Management for FinFET-Based Circuits Exploiting the Temperature Effect Inversion Phenomenon”. In: *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 2014, pp. 105–110.

[27] Sriram Sankar et al. “Datacenter Scale Evaluation of the Impact of Temperature on Hard Disk Drive Failures”. In: *ACM Transactions on Storage (TOS)* 9.2 (2013), p. 6.

[28] Minesh Patel et al. “Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices”. In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2019, pp. 13–25.

[29] A. T. Committee. *Data Center Power Equipment Thermal Guidelines and Best Practices Whitepaper*. https://tc0909.ashraetcs.org/documents/ASHRAE_TC0909_Power_White_Paper_22_June_2016_REVISED.pdf. 2016.

[30] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. “Power Capping: A Prelude to Power Shifting”. In: *Cluster Computing* 11.2 (June 2008), pp. 183–195. issn: 1386-7857. doi: 10.1007/s10586-007-0045-4.

[31] X. Wang and M. Chen. “Cluster-level Feedback Power Control for Performance Optimization”. In: *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. 2008, pp. 101–110. doi: 10.1109/HPCA.2008.4658631.

[32] R. Cochran et al. “Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps”. In: *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2011, pp. 175–185.

[33] Hiroshi Sasaki et al. “Characterization and Mitigation of Power Contention Across Multiprogrammed Workloads”. In: *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016. doi: 10.1109/iiswc.2016.7581266.

[34] Henry Hoffmann et al. “Application Heartbeats: a Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments”. In: *Proceedings of the 7th international conference on Autonomic computing*. ACM. 2010, pp. 79–88.

[35] Jackson Marusarz, Shannon Cepeda, and Ahmad Yasin. “How to Tune Applications Using a Top-Down Characterization of Microarchitectural Issues”. In: *Technical report*. Intel, 2013.

[36] Christos Kalogirou et al. “Exploiting CPU Voltage Margins to Increase the Profit of Cloud Infrastructure Providers”. In: *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2019, pp. 302–311.

[37] Abdellah Ouammou et al. “Modeling and Analysis of Quality of Service and Energy Consumption in Cloud Environment”. In: (2018).

[38] Abdellah Ouammou et al. “Energy Consumption and Cost Analysis for Data Centers with Workload Control”. In: *International Conference on Innovations in Bio-Inspired Computing and Applications*. Springer. 2017, pp. 92–101.

[39] Mohammad Wardat et al. “Cloud Data Centers Revenue Maximization Using Server Consolidation: Modeling and Evaluation”. In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2018, pp. 172–177.

[40] Sukhpal Singh Gill and Rajkumar Buyya. “A Taxonomy and Future Directions for Sustainable Cloud Computing: 360 Degree View”. In: *arXiv preprint arXiv:1712.02899* (2017).

[41] Anys Bacha and Radu Teodorescu. “Using ECC Feedback to Guide Voltage

Speculation in Low-Voltage Processors". In: *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society. 2014, pp. 306–318.

[42] Anys Bacha and Radu Teodorescu. "Dynamic Reduction of Voltage Margins by Leveraging on-chip ECC in Itanium II Processors". In: *ACM SIGARCH Computer Architecture News*. Vol. 41. 3. ACM. 2013, pp. 297–307.

[43] Yazhou Zu et al. "Adaptive Guardband Scheduling to Improve System-level Efficiency of the POWER7+". In: *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on*. IEEE. 2015, pp. 308–321.

[44] George Papadimitriou et al. "Voltage Margins Identification on Commercial x86-64 Multicore Microprocessors". In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2017. doi: 10.1109/iolts.2017.8046198.

[45] Vlasia Anagnostopoulou et al. "Power-Aware Resource Allocation for CPU- and Memory-Intense Internet Services". In: *International Workshop on Energy Efficient Data Centers*. Springer. 2012, pp. 69–80.

[46] Luiz André Barroso and Urs Hölzle. "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines". In: *Synthesis lectures on computer architecture* 4.1 (2009), pp. 1–108.

[47] Henry Hoffmann and Martina Maggio. "{PCP}: A Generalized Approach to Optimizing Performance Under Power Constraints through Resource Management". In: *11th International Conference on Autonomic Computing ({ICAC} 14)*. 2014, pp. 241–247.

[48] Martina Maggio et al. "Power Optimization in Embedded Systems via Feedback Control of Resource Allocation". In: *IEEE Transactions on Control Systems Technology* 21.1 (2011), pp. 239–246.

[49] David Meisner et al. "Power Management of Online Data-Intensive Services". In: *Proceedings of the 38th annual international symposium on Computer architecture*. 2011, pp. 319–330.

[50] Ripal Nathuji and Karsten Schwan. "Virtualpower: Coordinated Power Management in Virtualized Enterprise Systems". In: *ACM SIGOPS operating systems review* 41.6 (2007), pp. 265–278.

[51] Pavlos Petoumenos et al. "Power Capping: What Works, What Does Not". In: *ICPADS*. IEEE Computer Society, 2015, pp. 525–534.

[52] Rong Ge et al. “The Case for Cross-Component Power Coordination on Power Bounded Systems”. In: *Parallel Processing (ICPP), 2016 45th International Conference on*. IEEE. 2016, pp. 516–525.

[53] Jinsu Park, Seongbeom Park, and Woongki Baek. “RPPC: A Holistic Runtime System for Maximizing Performance Under Power Capping”. In: *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2018, pp. 41–50.

[54] Henry Hoffmann et al. “A Generalized Software Framework for Accurate and Efficient Management of Performance Goals”. In: *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*. IEEE. 2013, pp. 1–10.

[55] Andreas Merkel, Jan Stoess, and Frank Bellosa. “Resource-Conscious Scheduling for Energy Efficiency on Multicore Processors”. In: *Proceedings of the 5th European conference on Computer systems*. 2010, pp. 153–166.

[56] Mark Weiser et al. “Scheduling for Reduced CPU Energy”. In: *Mobile Computing*. Springer, 1994, pp. 449–471.

[57] Sergey Zhuravlev et al. “Survey of Energy-Cognizant Scheduling Techniques”. In: *IEEE Transactions on Parallel and Distributed Systems* 24.7 (2012), pp. 1447–1464.

[58] John L. Henning. “SPEC CPU2006 Benchmark Descriptions”. In: *SIGARCH Comput. Archit. News* 34.4 (Sept. 2006), pp. 1–17. issn: 0163-5964. doi: 10.1145/1186736.1186737. url: http://doi.acm.org/10.1145/1186736.1186737.

[59] Antoine Petitet Jack J. Dongarra Piotr Luszczek. “The LINPACK Benchmark: Past, Present and Future”. In: *Concurrency and Computation Practice and Experience* 10 (9 2003).

[60] G. Woltman and S Kurowski. *GIMPS, The Great Internet Mersenne Prime Search*. 2008. url: https://www.mersenne.org/.

[61] D. Hackenberg et al. “Introducing FIRESTARTER: A Processor Stress Test Utility”. In: *In Proceedings in the 4th Conference on International Green Computing (IGC)*. 2013, pp. 1–9. doi: 10.1109/IGCC.2013.6604507.

[62] *Stress-NG*. url: http://kernel.ubuntu.com/~cking/stress-ng/.

[63] A. Bacha and R. Teodorescu. “Using ECC Feedback to Guide Voltage Speculation in Low-Voltage Processors”. In: *In Proceedings of 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2014, pp. 306–318. doi: 10.1109/MICRO.2014.54.

[64] Vijay Janapa Reddi et al. “Voltage Emergency Prediction: Using Signatures to Reduce Operating Margins”. In: *In Proceedings of the 15th International Symposium on High Performance Computer Architecture, (HPCA)*. 2009, pp. 18–29.

[65] A. Liaw and M. Wiener. “Classification and Regression by RandomForest”. In: *R news* 2.3 (2002), pp. 18–22.

[66] *Perf: Linux Profiling with Performance Counters*. 2017. url: https://perf.wiki.kernel.org/index.php/Main_Page.

[67] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. “Estimating Mutual Information”. In: *Phys. Rev. E* 69 (6 2004), p. 066138. doi: 10.1103/PhysRevE.69.066138.

[68] Nathan Binkert et al. “The Gem5 Simulator”. In: *SIGARCH Comput. Archit. News* 39.2 (Aug. 2011), pp. 1–7. issn: 0163-5964. doi: 10.1145/2024716.2024718. url: http://doi.acm.org/10.1145/2024716.2024718.

[69] ArtForz. Jeff Garzik. *CPU-Miner*. url: https://github.com/tpruvot/cpuminer-multi.

[70] *Linux Kernel*. 2017. url: https://www.kernel.org/.

[71] L-N Pouchet. *PolyBench/C 3.2*. url: https://sourceforge.net/projects/polybench/.

[72] U. Mukhopadhyay et al. “A Brief Survey of Cryptocurrency Systems”. In: *In Proceedings of 14th Annual Conference on Privacy, Security and Trust (PST)*. 2016, pp. 745–752. doi: 10.1109/PST.2016.7906988.

[73] M. Kaliorakis et al. “Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions”. In: *IEEE Computer Architecture Letters* 17.2 (2018), pp. 109–112. issn: 1556-6056. doi: 10.1109/LCA.2018.2798604.

[74] G. Papadimitriou et al. “Voltage Margins Identification on Commercial x86-64 Multicore Microprocessors”. In: *In Proceedings of the 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2017, pp. 51–56. doi: 10.1109/IOLTS.2017.8046198.

[75] Anys Bacha and Radu Teodorescu. “Dynamic Reduction of Voltage Margins by Leveraging On-chip ECC in Itanium II Processors”. In: *SIGARCH Comput. Archit. News* 41.3 (June 2013), pp. 297–307. issn: 0163-5964. doi: 10.1145/2508148.2485948. url: http://doi.acm.org/10.1145/2508148.2485948.

[76] Y. Zu et al. “Adaptive Guardband Scheduling to Improve System-Level Efficiency of the POWER7”. In: *In Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2015, pp. 308–321.

[77] J. Leng et al. “Safe Limits on Voltage Reduction Efficiency in GPUs: A Direct Measurement Approach”. In: *In Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2015, pp. 294–307. doi: 10.1145/2830772.2830811.

[78] V. J. Reddi et al. “Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-Guided Thread Scheduling”. In: *In Proceedings of 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. 2010, pp. 77–88. doi: 10.1109/MICRO.2010.35.

[79] Andrew B. Kahng et al. “Designing a Processor from the Ground up to Allow Voltage/Reliability Tradeoffs”. In: *In Proceedings of the 16th International Conference on High-Performance Computer Architecture (HPCA)*. 2010, pp. 1–11.

[80] H. Duwe et al. “Rescuing Uncorrectable Fault Patterns in On-Chip Memories through Error Pattern Transformation”. In: *In Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*. 2016, pp. 634–644. doi: 10.1109/ISCA.2016.61.

[81] Zeshan Chishti et al. “Improving Cache Lifetime Reliability at Ultra-low Voltages”. In: *In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2009, pp. 89–99. isbn: 978-1-60558-798-1.

[82] C. Wilkerson et al. “Trading off Cache Capacity for Reliability to Enable Low Voltage Operation”. In: *In Proceedings of the 35th International Symposium on Computer Architecture (ISCA)*. 2008, pp. 203–214. doi: 10.1109/ISCA.2008.22.

[83] D. Ernst et al. “Razor: a Low-Power Pipeline based on Circuit-Level Timing Speculation”. In: *In Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2003.

[84] Smruti R. Sarangi et al. “EVAL: Utilizing Processors with Variation-induced Timing Errors”. In: *In Proceedings of the 41st Annual International Symposium on Microarchitecture (MICRO)*. 2008, pp. 423–434.

[85] Abhishek Tiwari, Smruti R. Sarangi, and Josep Torrellas. “ReCycle: Pipeline Adaptation to Tolerate Process Variation”. In: *In Proceedings of the 34th International Symposium on Computer Architecture, (ISCA)*. 2007, pp. 323–334.

[86] Xiaoyao Liang and David M. Brooks. “Mitigating the Impact of Process Variations on Processor Register Files and Execution Units”. In: *In Proceedings of the 39th Annual International Symposium on Microarchitecture (MICRO)*. 2006.

[87] N.S. Kim. *Resource and Core Scaling for Improving Performance of Power-Constrained Multicore Processors*. US Patent 9,606,842. 2017. url: https://www.google.com/patents/US9606842.

[88] P. J. Joseph, Kapil Vaswani, and M. J. Thazhuthaveetil. “Construction and Use of Linear Regression Models for Processor Performance Analysis”. In: *In Proceedings of the 12th International Symposium on High-Performance Computer Architecture (HPCA)*. 2006, pp. 99–108. doi: 10.1109/HPCA.2006.1598116.

[89] W. Jia, K. A. Shaw, and M. Martonosi. “Stargazer: Automated Regression-Based GPU Design Space Exploration”. In: *In Proceedings of the International Symposium on Performance Analysis of Systems Software (ISPASS)*. 2012, pp. 2–13. doi: 10.1109/ISPASS.2012.6189201.

[90] Benjamin C. Lee and David M. Brooks. “Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction”. In: *SIGPLAN Not.* 41.11 (Oct. 2006), pp. 185–194. issn: 0362-1340. doi: 10.1145/1168918.1168881. url: http://doi.acm.org/10.1145/1168918.1168881.

[91] Jangwoo Kim et al. “Modeling SRAM Failure Rates to enable Fast, Dense, Low-Power Caches”. In: *SELSE’09* (2009).

[92] H. Cherupalli, R. Kumar, and J. Sartori. “Exploiting Dynamic Timing Slack for Energy Efficiency in Ultra-Low-Power Embedded Systems”. In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 2016, pp. 671–681. doi: 10.1109/ISCA.2016.64.

[93] Jack Dongarra, Thomas Herault, and Yves Robert. “Fault Tolerance Techniques for High-Performance Computing”. In: *Fault-Tolerance Techniques for High-Performance Computing*. Springer, 2015, pp. 3–85.

[94] Thomas J Hacker, Fabian Romero, and Christopher D Carothers. “An Analysis of Clustered Failures on Large Supercomputing Systems”. In: *Journal of Parallel and Distributed Computing* 69.7 (2009), pp. 652–665.

[95] Yudan Liu et al. “An Optimal Checkpoint/Restart Model for a Large Scale High Performance Computing System”. In: *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE. 2008, pp. 1–9.

[96] B. Schroeder and G. A. Gibson. “A Large-Scale Study of Failures in High-Performance Computing Systems”. In: *International Conference on Dependable Systems and Networks (DSN’06)*. 2006, pp. 249–258. doi: 10.1109/DSN.2006.5.

[97] MIL-HDBK-338. *Military Handbook. Electronic Reliability Design Handbook*. 1998. url: https://www.weibull.com/knowledge/milhdbk.htm.

[98] Igor Bazovsky. *Reliability Theory and Practice*. Courier Corporation, 2004.

[99] Intel. *Reliability Report*. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/rr/rr.pdf. 2017.

[100] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. “CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management”. In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017.

[101] Youngtaek Kim and Lizy Kurian John. “Automated Di/Dt Stressmark Generation for Microprocessor Power Delivery Networks”. In: *In Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design (ISLPED)*. 2011, pp. 253–258. isbn: 978-1-61284-660-6.

[102] Y. Kim et al. “AUDIT: Stress Testing the Automatic Way”. In: *In Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2012, pp. 212–223.

[103] Zacharias Hadjilambrou et al. “Leveraging CPU Electromagnetic Emanations for Voltage Noise Characterization”. In: (2018).

[104] Teja Singh et al. “Zen: An Energy-Efficient High-Performance \×86 Core”. In: *IEEE Journal of Solid-State Circuits* 53.1 (2018), pp. 102–114.

[105] S. T. Kim et al. “Enabling Wide Autonomous DVFS in a 22 nm Graphics Execution Core Using a Digitally Controlled Fully Integrated Voltage Regulator”. In: *IEEE Journal of Solid-State Circuits* 51.1 (2016), pp. 18–30. issn: 0018-9200. doi: 10.1109/JSSC.2015.2457920.

[106] Henrique Madeira et al. “RIFLE: A General Purpose Pin-Level Fault Injector”. In: *Proc. of the European Dependable Computing Conference (EDCC)*. 1994.

[107] Jean Arlat et al. “Fault Injection for Dependability Validation: A Methodology and Some Applications”. In: *IEEE Trans. on Software Engineering* (1990).

[108] J. H. Barton et al. “Fault Injection Experiments Using FIAT”. In: *IEEE Trans. on Computers* (1990).

[109] Ghani A. Kanawati, Nasser A. Kanawati, and Jacob A. Abraham. “FERRARI: A Flexible Software-Based Fault and Error Injection System”. In: *IEEE Trans. Comput.* 44 (1995).

[110] Volkmar Sieh, Oliver Tschäche, and Frank Balbach. “VERIFY: Evaluation of Reliability Using VHDL-Models with Embedded Fault Descriptions”. In: *Proc. of the Symposium on Fault-Tolerant Computing (FTCS)*. 1997.

[111] Eric Jenn et al. “Fault Injection into VHDL Models: The MEFISTO Tool”. In: *Proc. of the Symposium on Fault-Tolerant Computing (FTCS)*. 1994.

[112] K. Parasyris et al. “GemFI: A Fault Injection Tool for Studying the Behavior of Applications on Unreliable Substrates”. In: *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP Int. Conference on*. 2014.

[113] Giorgis Georgakoudis et al. “REFINE: Realistic Fault Injection via Compiler-based Instrumentation for Accuracy, Portability and Speed”. In: *Proc. of the Int. Conference for High Performance Computing, Networking, Storage and Analysis*. 2017.

[114] Qining Lu et al. “LLFI: An Intermediate Code-Level Fault Injection Tool for Hardware Faults”. In: *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE. 2015, pp. 11–16.

[115] George Papadimitriou et al. “A System-Level Voltage/Frequency Scaling Characterization Framework for Multicore CPUs”. In: *13th IEEE Workshop on Silicon Errors in Logic-System Effects (SELSE ‘17). Boston, MA, USA*. 2017.

[116] Panos Koutsovasilis et al. “AcHEe: Evaluating Approximate Computing and Heterogeneity for Energy Efficiency”. In: *Parallel Computing* 73 (2018), pp. 52–67.

[117] Chris Lattner and Vikram Adve. “LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation”. In: *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*. CGO ’04. Palo Alto, California: IEEE Computer Society, 2004, pp. 75–. isbn: 0-7695-2102-9. url: http://dl.acm.org/citation.cfm?id=977395.977673.

[118] *Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory*. Tech. rep. LLNL-TR-490254. Livermore, CA, pp. 1–17.

[119] Henk Kaarle Versteeg and Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education, 2007.

[120] Mark L Wilkins, B Adler, et al. “Methods in Computational Physics”. In: *Calculation of elastic–plastic flow* (1964), pp. 211–263.

[121] Loup Verlet. “Computer ”Experiments” on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules”. In: *Phys. Rev.* 159 (1 1967), pp. 98–103. doi: 10.1103/PhysRev.159.98. url: http://link.aps.org/doi/10.1103/PhysRev.159.98.

[122] Lars Onsager. “Electric Moments of Molecules in Liquids”. In: *Journal of the American Chemical Society* 58.8 (1936), pp. 1486–1493.

[123] Ivo Babuška, Raúl Tempone, and Georgios E Zouraris. “Solving Elliptic Boundary Value Problems with Uncertain Coefficients by the Finite Element Method: the Stochastic Formulation”. In: *Computer methods in applied mechanics and engineering* 194.12 (2005), pp. 1251–1294.

[124] Luis J Roman and Marcus Sarkis. “Stochastic Galerkin Method for Elliptic SPDEs: A White Noise Approach”. In: *Discrete and Continuous Dynamical Systems-Series B* 6.4 (2006), p. 941.

[125] George Sarailidis and Manolis Vavalis. “On the Stochastic/Deterministic Numerical Solution of Composite Deterministic Elliptic PDE Problems”. In: *Proceedings of Mathematical Methods Computational Techniques in Science Engineering*. Bratislava, Slovakia, 2015.

[126] Nikolaos Bellas et al. “Real-time Fisheye Lens Distortion Correction Using Automatically Generated Streaming Accelerators”. In: *Proceedings of the 17th IEEE Symposium on Field Programmable Custom Computing Machines*. FCCM ’09. Napa, CA: IEEE Press, 2009, pp. 149 –156. isbn: 978-0-7695-3716-0. doi: 10.1109/FCCM.2009.16. url: http://doi.org/10.1109/FCCM.2009.16.

[127] Iain E. Richardson. *The H.264 Advanced Video Compression Standard*. 2nd. Wiley Publishing, 2010. isbn: 0470516925, 9780470516928.

[128] Philip H. S. Torr and Andrew Zisserman. “Feature Based Methods for Structure and Motion Estimation”. In: *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*. ICCV ’99. London, UK, UK: Springer-Verlag, 2000, pp. 278–294. isbn: 3-540-67973-1. url: http://dl.acm.org/citation.cfm?id=646271.685642.

[129] Heiko Hirschmüller. “Stereo Processing by Semiglobal Matching and Mutual Information”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 30.2 (2008), pp. 328–341.

[130] Koichiro Yamaguchi et al. “Continuous Markov Random Fields for Robust Stereo Estimation”. In: *Computer Vision–ECCV 2012*. Springer, 2012, pp. 45–58.

[131] Kazuhiro Yamaguchi, David McAllester, and Raquel Urtasun. “Robust Monocular Epipolar Flow Estimation”. In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pp. 1862–1869.

[132] NVIDIA. *NVML API Reference*. http://docs.nvidia.com/deploy/nvml-api/index.html. url: http://docs.nvidia.com/deploy/nvml-api/index.html.

[133] Intel. *Intel 64 and IA-32 Architectures Software Developer Manual*. Chapter 14.9.1. 2010.

[134] Shuai Che et al. “Rodinia: A Benchmark Suite for Heterogeneous Computing”. In: *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE. 2009, pp. 44–54.

[135] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[136] Hadi Esmaeilzadeh et al. “Neural Acceleration for General-Purpose Approximate Programs”. In: *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society. 2012, pp. 449–460.

[137] Amir Yazdanbakhsh et al. “Neural Acceleration for GPU Throughput Processors”. In: *Proceedings of the 48th International Symposium on Microarchitecture*. MICRO-48. Waikiki, Hawaii: ACM, 2015, pp. 482–493. isbn: 978-1-4503-4034-2. doi: 10.1145/2830772.2830810. url: http://doi.acm.org/10.1145/2830772.2830810.

[138] Stelios Sidiroglou-Douskos et al. “Managing Performance vs. Accuracy Trade-offs with Loop Perforation”. In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ESEC/FSE ’11. Szeged, Hungary: ACM, 2011, pp. 124–134. isbn: 978-1-4503-0443-6. doi: 10.1145/2025113.2025133. url: http://doi.acm.org/10.1145/2025113.2025133.

[139] Qian Zhang et al. “ApproxIt: An Approximate Computing Framework for Iterative Methods”. In: *Proceedings of the 51st Annual Design Automation Conference*. DAC ’14. San Francisco, CA, USA: ACM, 2014, 97:1–97:6. isbn: 978-1-4503-2730-5. doi: 10.1145/2593069.2593092. url: http://doi.acm.org/10.1145/2593069.2593092.

[140] Adrian Sampson et al. “EnerJ: Approximate Data Types for Safe and General Low-Power Computation”. In: *ACM SIGPLAN Notices*. Vol. 46. 6. ACM. 2011, pp. 164–174.

[141] Konstantinos Krommydas et al. “On the Characterization of OpenCL Dwarfs on Fixed and Reconfigurable Platforms”. In: *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*. IEEE. 2014, pp. 153–160.

[142] Anthony Danalis et al. “The Scalable Heterogeneous Computing (SHOC) Benchmark Suite”. In: *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM. 2010, pp. 63–74.